BAYESIAN EDGE ANALYTICS OF MACHINE PROCESS AND HEALTH STATUS IN AN IOT FRAMEWORK

A Dissertation
Presented to
The Academic Faculty

By

Daniel M Newman

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Mechanical Engineering

Georgia Institute of Technology

August 2020

BAYESIAN EDGE ANALYTICS OF MACHINE PROCESS AND HEALTH STATUS IN AN IOT FRAMEWORK

Approved by:

Dr. Thomas Kurfess, Advisor
Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. Christopher Saldana
Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. Shreyes Melkote
Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. Joshua Vaughan
Department of Mechanical Engineering
University of Louisiana at Lafayette

Dr. Al Salour
Boeing Research and Technology
The Boeing Company

Date Approved: July 7, 2020

It is not the critic who counts; not the man who points out how the strong man stumbles, or where the doer of deeds could have done them better. The credit belongs to the man who is actually in the arena, whose face is marred by dust and sweat and blood; who strives valiantly; who errs, who comes short again and again, because there is no effort without error and shortcoming; but who does actually strive to do the deeds; who knows great enthusiasms, the great devotions; who spends himself in a worthy cause; who at the best knows in the end the triumph of high achievement, and who at the worst, if he fails, at least fails while daring greatly, so that his place shall never be with those cold and timid souls who neither know victory nor defeat.

Theodore Roosevelt

To the helpers and mentors that made this achievement possible.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Kurfess for his mentorship throughout my doctoral studies. Thank you for asking me difficult questions, reassuring me when I was on the right path, and redirecting me when I was not. I would also like to thank my reading committee — Dr. Shreyes Melkote, Dr. Chris Saldana, Dr. Al Salour, and Dr. Josh Vaughan — for their feedback and support.

My doctoral studies would have been significantly less enjoyable and rewarding without having an amazing group of labmates to share graduate school with. I'm thankful to my fellow students who shared trials and triumphs with me over the past few years.

I would also like to thank teammates at The Boeing Company over my summer-long internship, namely Alex Burch and Eric Nicks, for helping me understand new concepts which were integral to the development of this thesis. Similarly, I'm grateful to Andrew Dugenske for his support in overseeing the development of several fundamental concepts in this work.

Lastly and most importantly, I'm grateful to my parents and girlfriend, Vallerie, for their unwavering support throughout my entire graduate studies. I could not have made it this far without your continuous confidence in me. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The past decade has seen an explosion in the capabilities of distributed computing, the Internet of Things, and open source software and hardware. As a result, internet connectivity has become ubiquitous across nearly all industries. Within the manufacturing sector, machine controllers now support protocols which make very detailed utilization information accessible. In addition, advances in embedded computing enable distributed, low-cost sensor deployment on an unprecedented scale. Leveraging these advances in data availability, this work presents a methodology for machine health monitoring in an Internet of Things architecture. Using modern messaging protocols, bidirectional communication between machine controllers and external sensors enables contextual data acquisition for tracking health trends in manufacturing equipment.

Using modern machine learning tools and embedded computing, a low-cost, integrated data acquisition platform is proposed in this work. Built on modern, open-source hardware and software, this platform enables high-quality sensor data acquisition and edge-based computation to facilitate machine health monitoring in an IoT framework. By leveraging proposed protocols for edge-based feature extraction, high-volume sensor data payloads are reduced in size to facilitate health monitoring and near real-time inference. The computational latency of this proposed methodology compares favorably to cloud-based solutions, where network transmission latency introduces significant variance in obtaining statistical features and model inference. A case study in tool wear analysis shows that CNC controller data may be used to contextualize accelerometer measurements and, in turn, facilitate training novelty detection and classification algorithms. These algorithms are then deployed to the edge device for near-real time inference.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Across a wide range of industries, significant investments are being made to monitor the health status of operation-critical systems. The economic driver behind this development is simple: unscheduled maintenance of critical assets carries significant costs [1]. Therefore, an ideal health monitoring system facilitates an operation in which machine health degradation can be diagnosed prior to failure, allowing corrective action to be taken. This approach is commonly referred to as predictive or condition-based maintenance [2, 3, 4].

Recent years have seen a substantial push towards remote system monitoring. Various methods have been proposed to utilize wireless networks and "Big Data" to create a connected, "smart" production factory [5]. Increasing efficiency and productivity through automation is another key thrust in this line of research. This overall effort is commonly referred to as Industry 4.0 [6, 7]. This framework requires the use of Cyber Physical Systems (CPS) — networked computing devices generally capable of data acquisition and transmission [8]. When these devices are connected via the internet, this network is generally referred to as part of the *Internet of Things* (IoT) [9].

To advance the state-of-the-art in manufacturing in the context of Industry 4.0, work must be done towards machine data acquisition and analytics to provide users meaningful, near real-time feedback on machine health and utilization. In large manufacturing facilities, hundreds of unique machines can be involved in production. The quantity, diversity, and complexity of manufacturing machines such as autoclaves and Computer Numerical Control (CNC) lathes and mills present a significant challenge from both a data acquisition and analysis standpoint. Many modern machines use digital controllers from which

relevant machine data may be accessed through standards such as MTConnect [10] or OPC-UA [11]. Other machines such as factory chilled water pumps may have no controller information readily accessible at all, but are equally vital to production.

1.2 Research Goals and Contributions

This work focuses on data 'connection' and 'conversion' in the context of CPS for manufacturing systems as shown in Figure 1.1 [12]. In the 'connection' layer, machines are connected to an IoT architecture so that relevant data may be extracted from them. The 'conversion' layer refers to the process by which raw data are converted into meaningful information such as health prognostics, utilization, and process anomaly detection. These two steps serve as the foundation for future development in manufacturing Industry 4.0. Before high-level collaboration and optimization may be undertaken from a production standpoint, reliable, useful information must be automatically extracted from the machines which are being monitored. This automatic information acquisition can be used to inform enterprise-level analytics engines while concurrently providing low-latency, machine-specific utilization and health metrics.

1.2.1 Thesis Contributions

The following chapters outline a digital framework for data acquisition, communication, and modeling to improve upon the current state-of-the-art in machine health and utilization monitoring in Industry 4.0.

1. Development of a digital architecture which facilitates contextual edge analytics – Chapter 3

   By concurrently recording data from both CNC machine controllers and external sensors, the health state of a manufacturing machine may be more completely captured. In this chapter, a strategy for integrating controller and

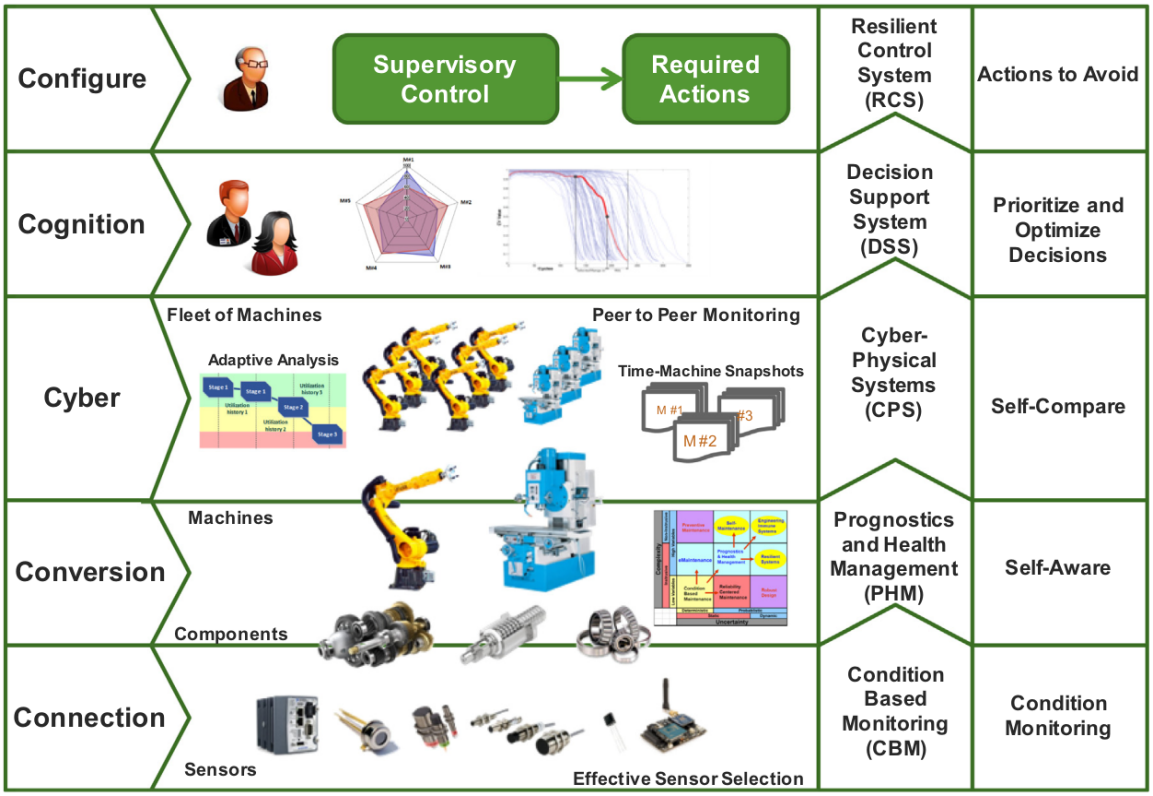


Figure 1.1: 5C Architecture for manufacturing CPS [12]

sensor measurements in an IoT framework is proposed. Basic demonstrations on a 3-axis mill are used to highlight long-term health monitoring applications.

2. Edge-deployable anomaly detection and classification algorithms – Chapter 4

   As modern machine learning and statistical classification tools become more powerful, they may also be deployed to low-power edge devices with exceptional efficiency. This chapter summarizes the deployment of state-of-the-art machine learning toolboxes and models to an embedded Linux computer and compares model performance across multiple alternatives. The results from this chapter reveal that even moderately complex neural networks may successfully be deployed to these edge devices with near real-time inference.

3. An edge device capable of controller and sensor data acquisition and analytics –

Chapter 5

In order to perform data acquisition and statistical inference in an IoT framework, an edge device which is capable of all of these tasks is needed. This chapter describes the design and implementation of an embedded Linux computer capable of high-quality sensor measurements, model inference, and IoT connectivity. This device is benchmarked against state-of-the-art cloud computing capabilities to compare worst-case message transmission latency.

4. An example deployment of this architecture and edge device for tool health monitoring in a 3-axis mill – Chapter 6

   Leveraging all of the methods proposed in the preceding chapters, a case study is undertaken in this chapter to investigate the efficacy of the proposed strategy in real-time health monitoring. Specifically, a statistical classifier is trained to detect when an excessively worn tool is used in an end-milling process.

CHAPTER 2
BACKGROUND

In this chapter, the background information necessary for this thesis are introduced. As this work focuses on advances in the Internet of Things and machine health monitoring, these topics comprise the majority of this chapter. Specifically, this chapter addresses the use of the internet of things within the context of the manufacturing industry. This discussion includes topics ranging from data acquisition methods and protocols to modern web messaging frameworks. These tools are used to facilitate machine health monitoring in a connected architecture. Therefore, this chapter also addresses the general history of machine health monitoring in the manufacturing industry. In the context of health monitoring, vibration analytics and frequency-domain techniques have a rich background and broad applicability across a range of machines and industries. In addition, vibration analysis serves as an excellent benchmark for high-density data which must be properly managed in an IoT framework. For these reasons, a special focus is given to vibration and frequency-domain analysis in this work.

2.1 Industrial Internet of Things

The Internet of Things promises to facilitate the connections between devices on an unprecedented scale [13, 14]. Recently, a substantial research effort has been invested into the application of IoT technologies in the context of manufacturing [15]. Within this framework, manufacturing equipment transmit data to an enterprise-level or cloud network, where data mining is used to extract information to improve the efficiency of the manufacturing process. This application of IoT is commonly referred to as the Industrial Internet of Things [7]. Within the IIoT, wireless networks play a major role [16]. As such, network latency, capacity, flexibility, and scalability are all major points of concern. These

networks have been used to facilitate both real-time and historical data analysis [17].

The broad, interdisciplinary nature of the industrial Internet of Things has led to a vast quantity of research aimed at addressing its various aspects. At the highest level of abstraction, strategies have been proposed to define the connection of Cyber-Physical Systems for enterprise-level decision making [18]. To implement IIoT strategies, it is crucial to develop architectures which facilitate the connectivity of devices. To this end, a substantial research effort has been directed toward specifying IIoT architectures [19, 20]. These architectures rely on devices such as factory machines and sensors to transmit relevant data. With access to these data sources through the IIoT architecture, analytics such as predictive maintenance and machine utilization can be performed.

The ultimate objective of the Industrial Internet of Things is a manufacturing enterprise which rapidly adapts to changes in market while minimizing waste and improving product quality [8, 21]. To accomplish this objective, IIoT leverages advances in embedded computing, Cyber Physical Systems, cloud computing, big data, and internet connectivity [22]. To properly assemble a framework for utilizing each of these tools, it is vital to develop a logical, cognitive structure which associates each tool with a purpose within this framework [12, 23].

A typical representation of the Industrial Internet of Things is shown in Figure 2.1. At the bottom of the image, physical devices such as CNC machines and sensors supply the low-level data from factory equipment. These devices are connected to an industrial network for data transmission across the enterprise. A slew of possible connectivity options including various wired and wireless protocols are available for this task, where wireless options are commonly used due to their convenience. At this level, it is also important to consider a standardized message structure for the data being sent across the network. Modern formats such as XML and JSON are useful for this task. From the industrial network, data are typically sent to a cloud platform where storage and computing resources are available. These systems track historical data and perform prognostics on machine

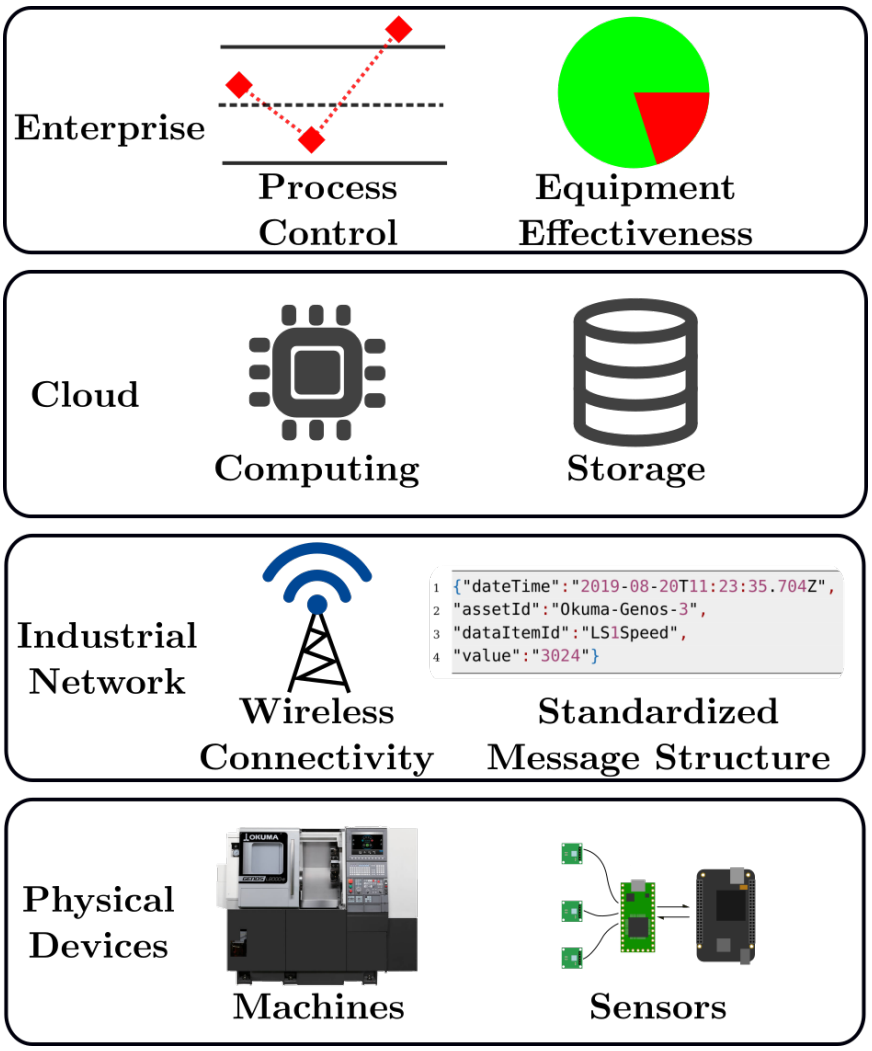


Figure 2.1: Industrial Internet of Things Structure

health. Finally, these data are used to inform enterprise-level metrics such as statistical process control and Overall Equipment Effectiveness (OEE).

2.2 Messaging Protocols

When developing an IoT architecture, it is critical to choose an appropriate communication protocol for transmitting data. Two such protocols are Hypertext Transfer Protocol (HTTP) and Message Queuing Telemetry Transport (MQTT). The fundamental distinction between these protocols is that MQTT uses a publish/subscribe format while HTTP uses a request/response format. As a result of this difference, MQTT has lower overhead for the

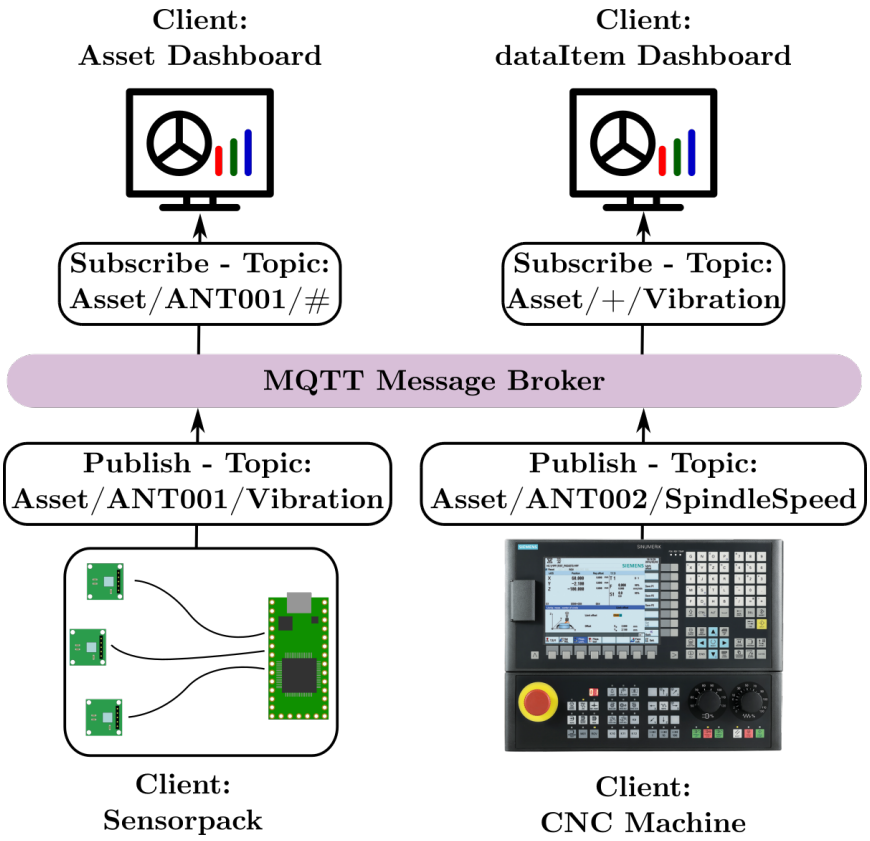


Figure 2.2: MQTT Example publish-subscribe implementation

continuous data streaming requirements in an IoT framework [24]. While newer protocols such as Constrained Application Protocol (CoAP) may be technically more efficient, MQTT is currently an accepted standard for IoT applications and is widely supported. Still, HTTP is invaluable in the modern internet and is the basis for many applications.

A simple MQTT implementation example is shown in Figure 2.2. Central to the MQTT protocol is the Message Broker, which routes all incoming and outgoing messages. Various clients can connect to the Message Broker in order to publish and subscribe to desired topics. The topic structure is also shown in this figure. For instance, the sensor pack may publish data to *Asset/ANT001/Vibration*. In this topic structure, the second layer, *ANT001*, specifies the asset name and the third layer, *Vibration*, specifies the data item ID. If a client wishes to subscribe to all messages pertaining to Asset *ANT001*, they may subscribe to the topic *Asset/ANT001/#* as in the upper left client example. The "#" symbol indicates a

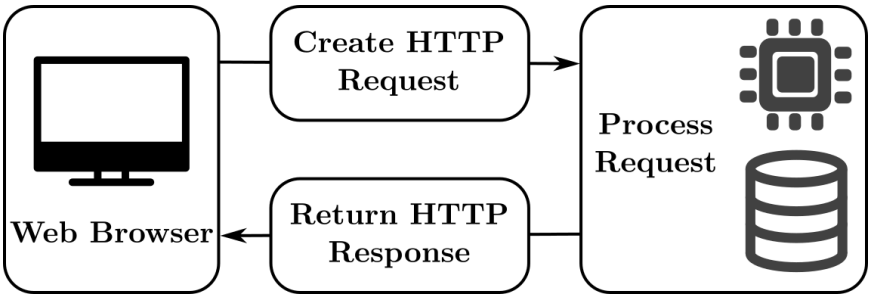


Figure 2.3: HTTP example

wildcard, meaning that the client will catch all messages with the preceding topic structure. Analogously, if a client wishes to subscribe to all vibration data regardless of source, they may subscribe to the topic, *Asset/+/Vibration*, as in the upper right example. Here, the "+" symbol indicates a single-level wildcard. This subscription structure will capture vibration data from all assets.

Unlike MQTT, HTTP is a request-response protocol, as demonstrated in Figure 2.3. In an HTTP architecture, a server is dedicated to handle requests as sent from clients. The nature of the request may vary from simply returning a desired value to performing a statistical calculation and anything in between. In this protocol, the computational load of processing requests is centrally located. To properly scale this framework, the server must be scaled in computational power to accommodate larger numbers of web clients. This need for large-scale central processing can become expensive in a distributed environment such as the IoT. In addition, the request-response framework results in higher latency and greater overhead in systems which regularly exchange information.

2.2.1 Data formats

While messaging standards define the means by which messages are sent and received in a modern web framework, they rely on existing data formats to make messages easily parsed. One such format is the Extensible Markup Language (XML), which originated in 1998 [25]. This language is widely used as the basis for word processor document storage as well as standards such as MTConnect. XML provides a framework wherein

```
<worksheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
 xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships">
  <sheetPr filterMode="false">...</sheetPr>
  <dimension ref="A1:P121"/>
  <sheetViews>...</sheetViews>
  <sheetFormatPr defaultRowHeight="13.8" zeroHeight="false" outlineLevelRow="0" outlineLevelCol="0"/>
  <cols>...</cols>
  <sheetData>...</sheetData>
  <printOptions headings="false" gridLines="false" gridLinesSet="true" horizontalCentered="false"
  verticalCentered="false"/>
  <pageMargins left="0.7" right="0.7" top="0.75" bottom="0.75" header="0.511805555555555"
  footer="0.511805555555555"/>
  <pageSetup paperSize="1" scale="100" firstPageNumber="0" fitToWidth="1" fitToHeight="1" pageOrder="downThenOver"
  orientation="portrait" blackAndWhite="false" draft="false" cellComments="none" useFirstPageNumber="false"
  horizontalDpi="300" verticalDpi="300" copies="1"/>
  <headerFooter differentFirst="false" differentOddEven="false">...</headerFooter>
</worksheet>
```

Figure 2.4: XML example code

```
{
    "Mean":-0.01503215,
    "Vibration":[-0.04808,-0.04539,...,-0.04703]
}
```

Figure 2.5: Example JSON message

complex document structures can be created in a standardized way, provided that standard is documented and maintained. As an example of XML's ubiquity and power, Microsoft Excel spreadsheets are stored in an XML structure. A very simple spreadsheet is highlighted in Figure 2.4.

Javascript Object Notation (JSON) is a modern data format which grew out of the need for standardized data transmission in an internet framework [26]. With a simple syntax and limited number of data types, JSON is lightweight and intuitive to use. As a result, messages sent through protocols such as HTTP or MQTT are often formatted as JSON strings so that an interpreter may easily parse their contents for use in a web application. An example JSON message is shown in Figure 2.5. The message is bracketed to define its start and end points. Different keys are given in double quotes with their values separated by a colon. In this structure, specific data items are easily interpreted by human users and machines. This simplicity and user-readability is a significant benefit of JSON over XML.

Figure 2.6: Modern CNC controller [32]

2.3 Data Sources and Protocols

Fundamental to the Industrial Internet of Things, a variety of machines and sensors constitute the modern factory. Many modern machines come equipped with advanced controllers such as the Siemens 828D pictured in Figure 2.6. These controllers enable data acquisition through protocols such as MTConnect or OPC-UA.

Recent work has been applied toward developing IoT frameworks specifically aimed at transmitting this data from CNC machines [27]. Later work used MTConnect data, along with part and process information to monitor a manufacturing process [28]. This work also discussed the development of an IoT framework built on HTTP and designed to leverage cloud computing. Recently, MTConnect and OPC-UA data have been combined with sensor data in a local cloud to generate machine learning algorithms for predictive maintenance [29]. Machine and production data have also been aggregated in order to visualize power consumption [30]. Latency comparisons for machine learning algorithms on equipment data have been performed for local and remote cloud architectures [31]. In recent years, work has been done to demonstrate the use of IoT methodologies in

conjunction with sensor data acquisition. For example, a low-power data acquisition device has been designed to monitor the vibrations of equipment and transmit them for analytics [33]. Similar work demonstrated the use of low-power electronics to develop a low-cost wireless accelerometer platform [34, 35]. This low-power sensor approach has also been investigated specifically for the Field-Programmable Gate Array (FPGA) chip architecture [36]. Wireless sensors have also been deployed in an automated irrigation system [37]. Lastly, onboard classification algorithms in embedded devices have been shown to be feasible and more energy efficient than raw data transmission for accelerometer measurements [38].

2.3.1 MTConnect

MTConnect is a read-only, XML-based standard for accessing and organizing machine data [10]. Many modern CNC machine manufacturers such as Okuma and Mazak support MTConnect as the protocol for retrieving controller data from their equipment. A typical, simplified implementation of this standard is shown in Figure 2.7. A machine generates data such as coordinate axis locations, motor power consumption, and run time. These data are mapped to an MTConnect Adapter. Because the Adapter is machine-specific, it is typically designed and supplied by the machine manufacturer. From this point, an MTConnect Agent runs a local web server which makes the machine data available on a local network through HTTP requests. Unlike the Adapter, the MTConnect Agent is more generally designed to interface with any Adapter. This generalized interface ensures standardization of the MTConnect protocol across devices. MTConnect specifies an API for accessing these data. When the client requests a sample from the Agent, an XML response is sent which contains data such as *dateTimes* and *dataItemIds*.

In addition to providing an endpoint for machine controller data, MTConnect also defines a standard and framework for accessing manufacturing equipment data. Within the MTConnect standard, data items have specific definitions, units, and formats. As an

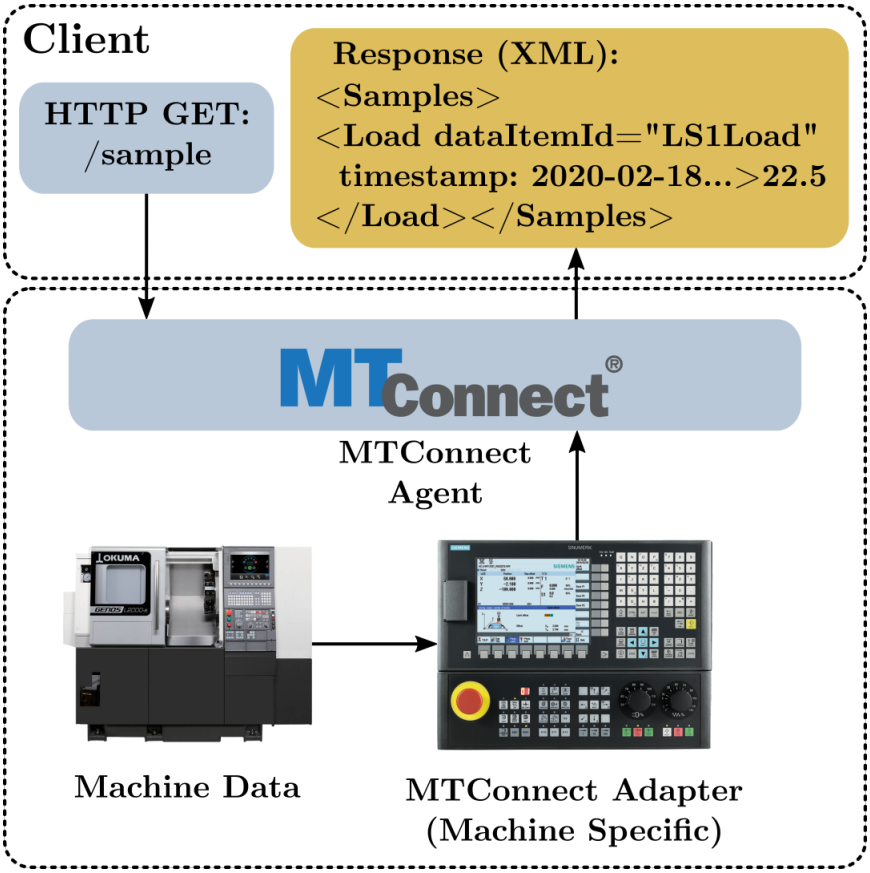


Figure 2.7: MTConnect typical implementation

example, consider a sample from the NIST Smart Manufacturing System Testbed [39], shown in Figure 2.8. This figure shows a highly truncated XML response from the testbed server and shows a single sample, `GFAgie01-C_2`. This *dataItemId* has a name — `Cposition` — and a component — `Rotary` — indicating that this is a positional data tag of a rotary axis. By definition in MTConnect standards, the value transmitted by this data item is in units of degrees. The *subType* `ACTUAL` indicates that this value represents the real value of the axis, rather than the commanded position. All of these elements are part of the MTConnect standard.

2.3.2 OPC-UA

Unlike MTConnect, OPC-UA is a protocol for accessing machine data but does not specify an organizational standard for presenting said data. In addition, OPC-UA is a read-write

```
<Streams>
<DeviceStream name="GFAgie01" uuid="mtc_adapter001">
<ComponentStream component="Rotary" name="C"
    componentId="GFAgie01-C_1">
<Samples>
<Angle dataItemId="GFAgie01-C_2"
    timestamp="2020-04-22T09:18:17.165304"
    name="Cposition" sequence="399615523"
    subType="ACTUAL">359.9999</Angle>
</Samples>
</ComponentStream>
</DeviceStream>
</Streams>
```

Figure 2.8: MTConnect sample example

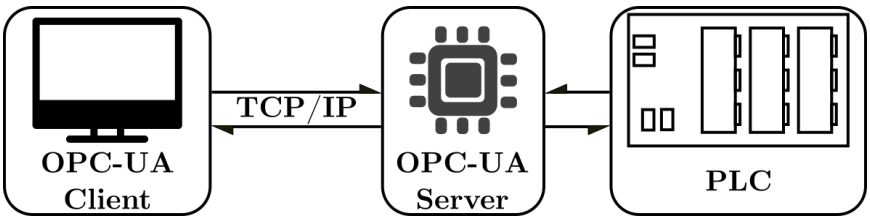


Figure 2.9: OPC-UA typical implementation

standard where values on a machine may be modified by a client with proper permissions. The basic components in OPC-UA are shown in Figure 2.9. A client computer connects with the OPC-UA server via the TCP/IP protocol to read and write values. The OPC-UA server in turn reads and writes data from the PLC, which allows for low-level parameter visibility.

Recently, the OPC Foundation and MTConnect have released standards for unifying these frameworks [40]. By adopting and publishing a uniform standard, interoperability between MTConnect-native and OPC-UA-native machines in an enterprise may be achieved. For example, a machine with an MTConnect adapter may have an OPC-UA server installed to interface with the machine.

2.3.3 Sensor Data

With advances in embedded computing, modern sensors can be readily incorporated into an IIoT architecture for machine health monitoring. Historically, most sensor data acquisition for studies such as vibration analytics has been done using highly accurate, expensive devices such as the Analog-to-Digital DAQ from National Instruments in Figure 2.10. These data acquisition platforms can sample analog sensors at sampling rates greater than 20kHz and are coupled with sophisticated signal processing software such as LabView to perform advanced analysis on captured signals. While this platform provides high quality sensor data, a number of factors inhibit its use in an IoT framework. First, the DAQ typically requires 120VAC input to accommodate the high-voltage ($\geq$ 24VDC) sensors and data acquisition unit. Standard 120 Volt outlets are often inconveniently located relative to production equipment, making it difficult to permanently install such a device without introducing safety concerns from Foreign Object Debris. While other methods, such as tapping into power supplies intended for the production equipment, are technically possible, they also raise safety questions or may unnecessarily interfere with production-critical equipment. As a result, this power supply requirement brings significant limitations to distributed deployment. Second, this platform relies on licensed software. To distribute and analyze data streams from multiple machines, several licenses must be purchased and managed. This raises other issues, such as proprietary data formats which may not be readily transferred to other software. As a result of the high-quality, high-frequency sampling that this platform can perform, large quantities of data must be processed for each sensor through this proprietary software.

Recently, companies have been leveraging the advances in embedded computing to build and market IoT-compatible sensor kits at costs well below those of traditional platforms. For example, National Control Devices offers pre-built and custom options for health monitoring and IoT applications such as the wireless kit shown in Figure 2.11. While these products represent a substantial advancement in the availability of low-cost,

Figure 2.10: Analog-to-Digital DAQ from National Instruments [41]

low-power data acquisition and transmission, they typically lack data processing capabilities. In addition, they are not configured to transmit high-density vibration signals. Instead, they typically send discrete measurements. Lastly, they are not configured to transmit data in a standardized manner. With these weaknesses, a significant gap exists in edge data acquisition devices which are capable of advanced analytics and standardized data transmission in an IoT framework.

2.4 Machine Health Monitoring

Machine health monitoring is an enormous field of study which touches a large range of industries [43]. The objectives of health monitoring are to diagnose existing issues in industrial equipment and predict remaining useful life in production-critical assets. Although a significant amount of work in Industrial IoT has been applied to equipment monitoring, the bulk of the research on equipment monitoring has been performed without consideration for data transmission in a modern framework. For this reason, it is important

Figure 2.11: IoT sensor kit from National Control Devices [42]

to discuss the broad history of machine health monitoring and the methods used to perform the necessary analytics. Specifically, this review focuses on the application of machine health monitoring to vibrating systems such as rolling element bearings. Due to their ubiquity in manufacturing equipment and their importance in transmitting rotational mechanical energy, bearing analysis has been and will continue to be a significant focal point in the context of machine health monitoring research [44].

Historically, vibration analytics have been performed through signal processing methods [45, 46, 47, 48]. Typically, a vibration measurement device such as an accelerometer or acoustic emission sensor is placed on the housing of a rotating piece of machinery for data acquisition. Analytics on the resulting signal tend to reveal possible defects and end-of-life indicators.

In one major area of study, statistical and machine learning methods have been employed to detect defects in vibrating machinery from acoustic emission and accelerometer data. For instance, wavelet transforms have been used as a feature extraction tool for bearing defect detection using constrained optimization [49]. With seeded faults, various time-series statistics such as kurtosis, maximum amplitude, and root mean square

have been compared using acoustic emission sensors and accelerometers [50]. In a similar application, mathematical models have been applied to vibration measurements to approximate defect size [51]. Because some bearing defects tend to exhibit impulsive characteristics, spectral kurtosis has been shown to be promising in detecting these faults [52].

Another significant area of study involves attempting to determine the remaining useful life of rotating elements such as bearings from vibration and acoustic emission data. Combining several time and frequency domain statistics from vibration data, a statistical model has been developed to approximate remaining useful life of bearings [53]. In other work, nonlinear feature extraction methods were employed to train a support vector regression model to perform the same task [54].

2.5 Frequency-Domain Signal Processing

For rotating machinery such as motors and pumps, sampled vibration signals can be used to extract frequency-domain information. In many cases, specific frequencies can be associated with different failure modes or health metrics. By evaluating the relative prominence of these frequencies in a vibration signal, it is possible to determine the health status of such vibrating equipment.

The Fourier Transform is a vital tool in frequency-domain analysis. For a discrete-time signal, the Fourier Transform can be defined by

$$X(\omega) = \sum_{n=-\infty}^{\infty} x_n e^{-j\omega n}, \tag{2.1}$$

where $X(\omega)$ is a function of frequency that has been extracted from the time-domain signal. In theory, this transform can be used to define the frequency content of a signal exactly. In practice, a number of factors make it difficult to glean meaningful information from a real vibration signal using (2.1) directly. First, this equation yields a complex-valued

function, $X(\omega)$. Because physical signals have no complex component, this abstraction does not readily follow intuition. Second, this equation requires an infinite summation of the time-series data. This is obviously impossible for a real signal, where high-frequency vibration data are often sampled in increments on the order of seconds. Lastly, this transform is not robust to the presence of noise in the vibration signal.

To address the aforementioned issues, (2.1) is first modified to account for the limited number of samples, N

$$X[n] = \sum_{k=0}^{N-1} x[k] e^{-j\frac{2\pi}{N}nk}, \tag{2.2}$$

which can be represented as matrix multiplication

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \dots & W^0 \\ W^0 & W^1 & W^2 & \dots & W^{N-1} \\ W^0 & W^2 & W^4 & \dots & W^{2(N-1)} \\ \vdots & & & & \\ W^0 & W^{N-1} & W^{2(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} \tag{2.3}$$

or, more concisely

$$X = \mathcal{W}x, \tag{2.4}$$

where $W = e^{-j\frac{2\pi}{N}}$. In the preceding equations, X is the Fourier Transform of the time series signal x, sampled at discrete intervals. For this reason, (2.4) is called the Discrete Fourier Transform (DFT). In practice, this is often interchanged with the Fast Fourier Transform (FFT), which is simply a computationally efficient method of implementing the DFT. The units of X are expressed in Hertz. Importantly, this raw mathematical form contains frequencies on the interval

$$-\frac{F_s}{2} \leq F \leq \frac{F_s}{2}, \tag{2.5}$$

where $\frac{F_s}{2}$ is the Nyquist frequency, and F is the array of frequencies associated with X. The resulting frequency resolution of F is therefore

$$\Delta f = \frac{F_s}{N}, \tag{2.6}$$

where Δf provides the number of Hertz between elements in F. The inclusion of negative frequencies in (2.6) provides a convenient mathematical transformation, but does not yield meaningful diagnostic information. For this reason, it is common to ignore the negative frequency components of the Fourier Transform when performing analysis. Finally, it is typical in signal processing applications to represent the Fourier Transform magnitudes, X, by element-wise multiplication with their complex conjugates. This operation results in a loss of phase information from the DFT, but allows for a clearer representation of the magnitudes of the frequencies present in the signal.

As an exercise in illustrating the Discrete Fourier Transform, consider a signal

$$y(t) = \sum_{i=0}^{M} A_i \sin(2\pi\omega_i) + \nu + A_{off}, \tag{2.7}$$

where A_i and ω_i are the i^{th} amplitude and frequencies, respectively. In addition, $\nu \sim \mathcal{N}(0, \sigma)$ represents white noise and A_{off} is some DC offset. For simplicity, first consider a signal with $A = [1, 1, 1]$, $\Omega = [256, 272, 300]$, $\sigma = 0$, and $A_{off} = 0$. This signal is sampled at $F_s = 1024$ Hz where $N = 128$ sample points.

The sampled signal is shown in Figure 2.12. To illustrate sampling rate limitations in accurately capturing a signal, the "true" signal is overlaid with the sampled signal in Figure 2.12. Although the frequencies of these signals are below the Nyquist frequency, it is apparent that the peaks of the true signal are not fully captured.

The unmodified DFT as determined by (2.2) is shown in Figure 2.13a. As mentioned, this expression of the Fourier Transform contains a real and imaginary component in addition to negative frequencies. Although it is apparent from this figure that three

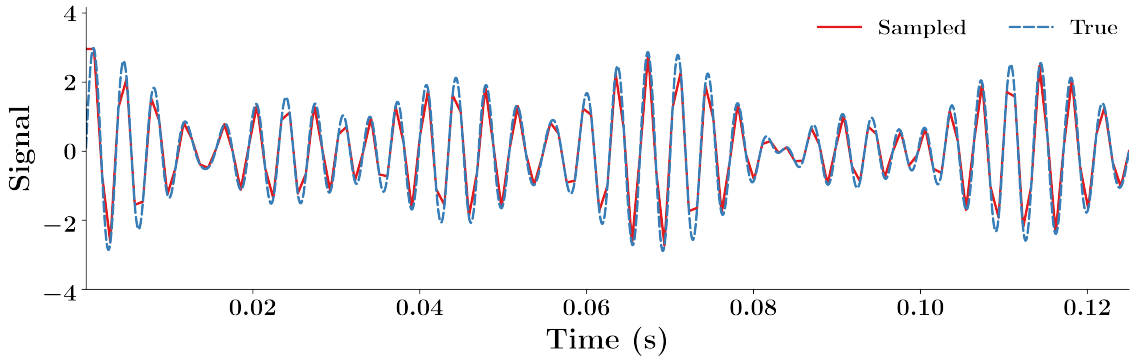
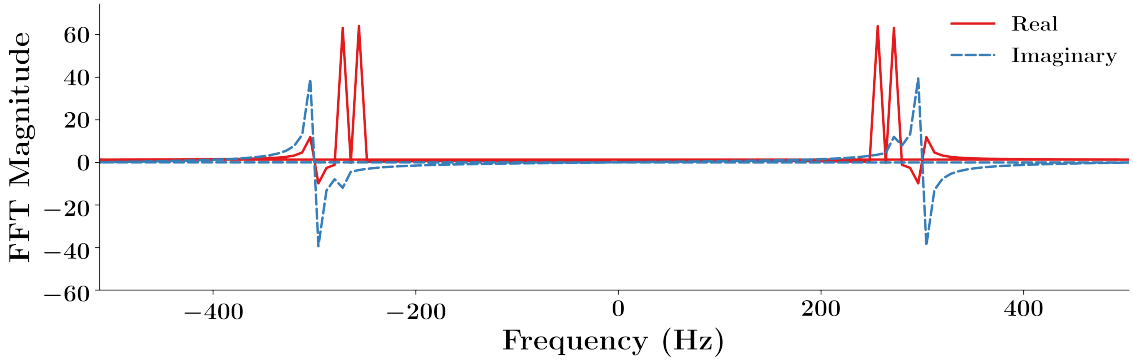
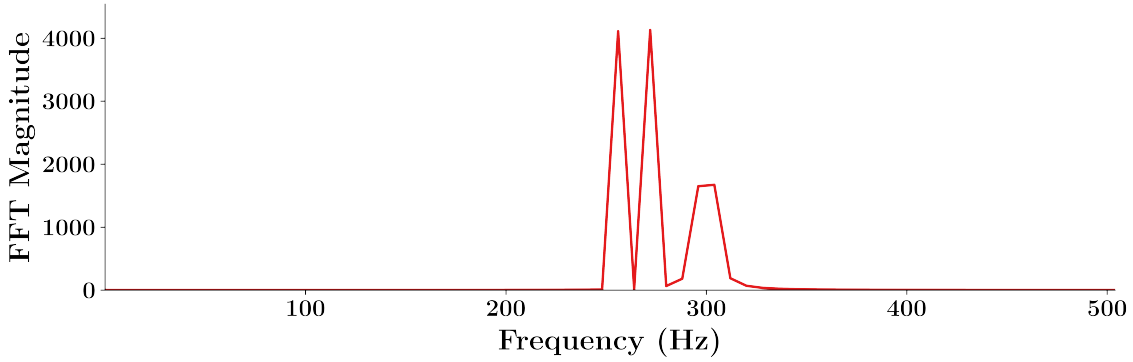


Figure 2.12: Sampled and true signal



(a) Raw DFT



(b) DFT with readability modifications

Figure 2.13: Example signal with no noise - DFT

positive frequencies are present in the example signal, this raw mathematical form contains extraneous information which have no physical meaning or easily applied intuition. By ignoring the negative frequencies and taking the magnitudes of the complex-valued transform, as shown in Figure 2.13b, the signal frequency content becomes significantly

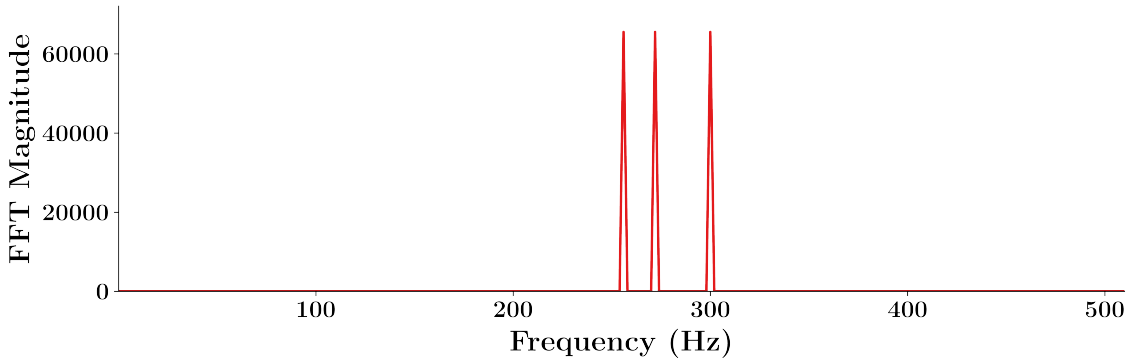


Figure 2.14: DFT with increased sampling duration

easier to interpret. Here, it is obvious that the sampled signal appears to be composed of three frequencies with different amplitudes. Because the ground truth — all amplitudes are identical — is known, it is clear that the relatively low sampling rate failed to accurately capture these signals.

2.5.1 Improving DFT quality through sampling

Multiple techniques and methods can be employed to improve the quality of frequency content extracted from a discretely sampled signal. For instance, it is easy to demonstrate the effectiveness of increasing the number of data samples, whether by increasing the sampling rate or the sampling duration.

When the example signal is sampled at the same F_s, but over a longer duration of 0.5 second, the total number of sample points increases to $N = 512$. The resulting DFT is shown in Figure 2.14. By increasing the record length, the DFT now contains 257 points (including DC). Clearly, this has substantially improved the resolution compared to Figure 2.13b. In addition, it is clearer from this figure that the amplitudes of each frequency are equal.

If the original number of sample points is held constant but F_s is increased, the frequencies of interest become lower relative to the Nyquist frequency. Consider the case where F_s is increased by a factor of two to $F_s = 2048$. The resulting DFT is shown in

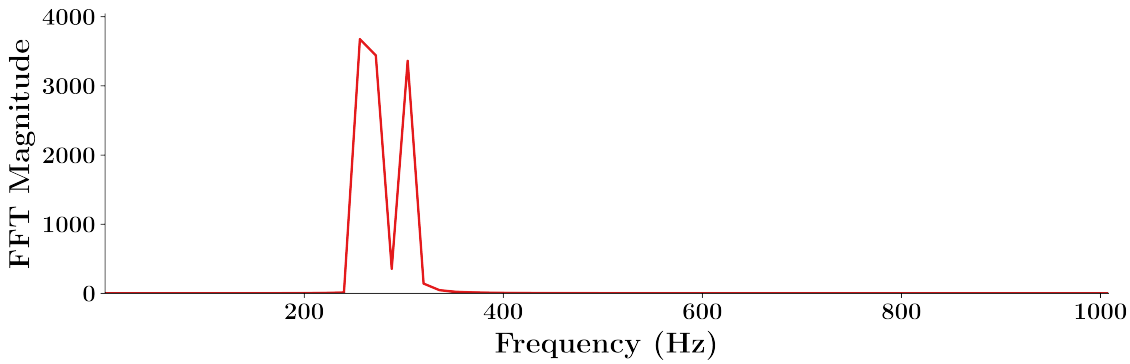


Figure 2.15: DFT with increased F_s

Figure 2.15. In this case, the high-frequency peak is now more clearly captured, but the two low-frequency peaks have become muddled together. In this example, the DFT has the same number of data points as in Figure 2.13b, but spans a larger range of frequencies due to the higher sampling rate. As a result, the Δf between frequency bins is larger, meaning a loss in resolution in frequency content which causes this "squishing" together of frequencies.

Clearly, a trade-off exists between sampling rate and record length. While increasing the sampling rate allows for capturing higher frequencies, it also reduces the resolution of the DFT if the record length is not also increased. While it may seem obvious to sample "as fast as possible" and store as many data points as possible to overcome resolution and sampling issues, it is important to identify a targeted frequency resolution and the frequencies of interest.

2.5.2 Avoiding Aliasing

When sampling vibration data, it is important to avoid aliasing the signal. Aliasing occurs when the sampling rate is not sufficiently high to capture all of the vibration present in the system. If this happens, the excessively high frequencies will interfere with the rest of the signal.

As an example, consider the previous signal, where the high-frequency component is

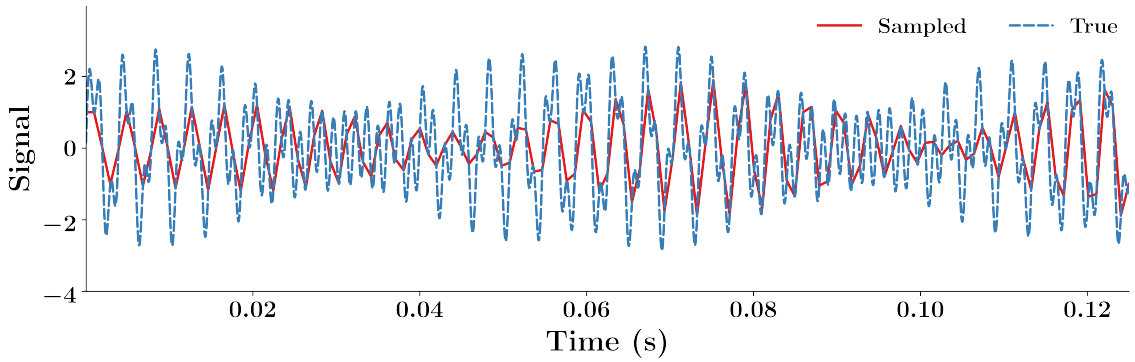


Figure 2.16: Aliased signal

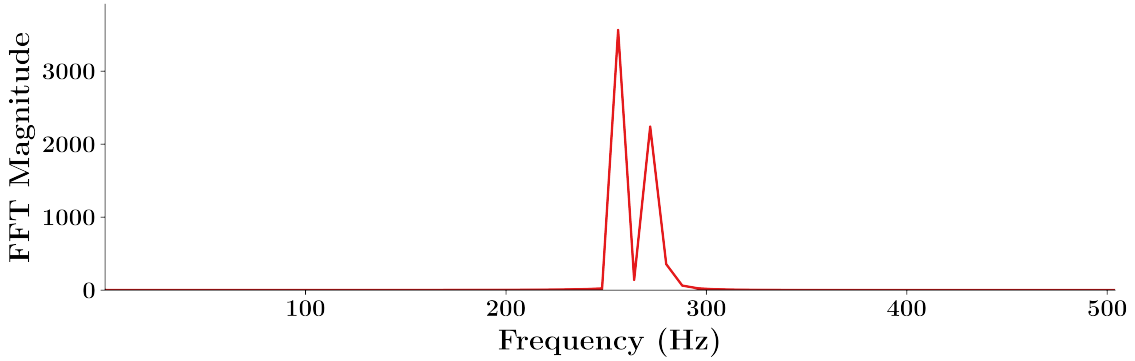


Figure 2.17: DFT of the aliased signal

shifted to 750Hz. This component is significantly above the Nyquist frequency of 512Hz. Figure 2.16 illustrates the destructive nature of aliasing. Clearly, the sampled signal does not resemble the original signal whatsoever. This is further illustrated by examining the DFT in Figure 2.17. In this specific example, the aliased signal appears to only contain the two lower frequencies, with one amplitude higher than the other.

From this example, it is clear that aliasing can confound a sampled signal. With arbitrary high-frequency content, an aliased signal can take any form. For this reason, it is important to sample at a sufficiently high frequency to capture the entire signal without aliasing. It is also helpful to place a low-pass filter on the sampling device to reduce frequencies above a desired setpoint.

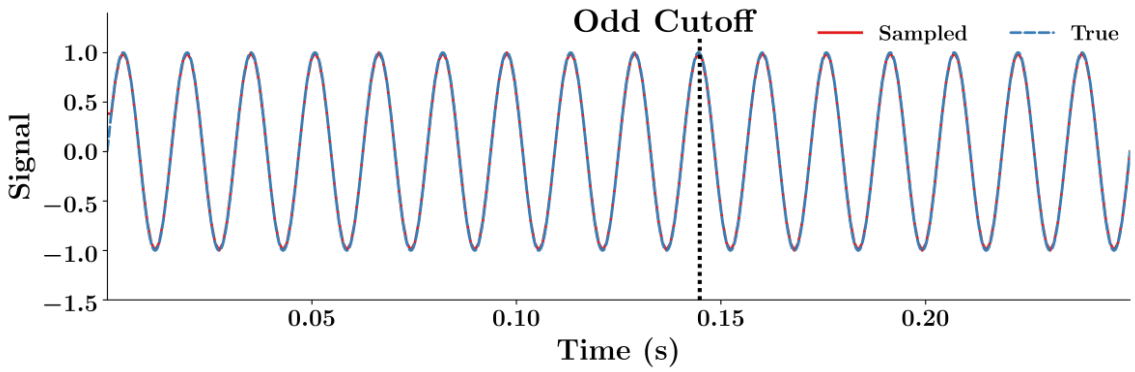


Figure 2.18: Full signal and non-integer cutoff

2.5.3 Reducing Spectral Leakage

Another important consideration when extracting the frequency content of a signal is minimizing "spectral leakage." This phenomenon occurs when a sample captures a non-integer number of oscillations for the signal under investigation. Spectral leakage can also occur when the signal has a non-zero mean.

For illustration, consider a signal which contains a single frequency at $\omega = 64$Hz, and an amplitude of 1. Again, the sampling rate for capturing this signal is 1024Hz, and the record length is $N = 256$. This signal is shown in Figure 2.18. Note that the output of the signal for this full record begins and ends at $y = 0$. This is an important nuance of the DFT; if a signal of zero mean is captured in this way, the DFT will exhibit a single peak at the appropriate frequency. However, if the record is cutoff at some arbitrary point where the signal is nonzero (such as the line labeled "odd cutoff" in this figure), spectral leakage will occur. Instead of a single peak, the entire DFT will be corrupted by broadband noise. This is also true for a signal with non-zero mean. If this signal has a steady-state offset, a "ski-slope" effect will be apparent in the DFT, where a large peak at DC will slowly ramp down at high frequencies.

This corruption is shown in Figure 2.19. The y-axis in this figure is shown on a logarithmic scale to clearly show how much broadband noise is introduced by capturing a non-integer number of periods for the signal in question. Where the DFT from the full

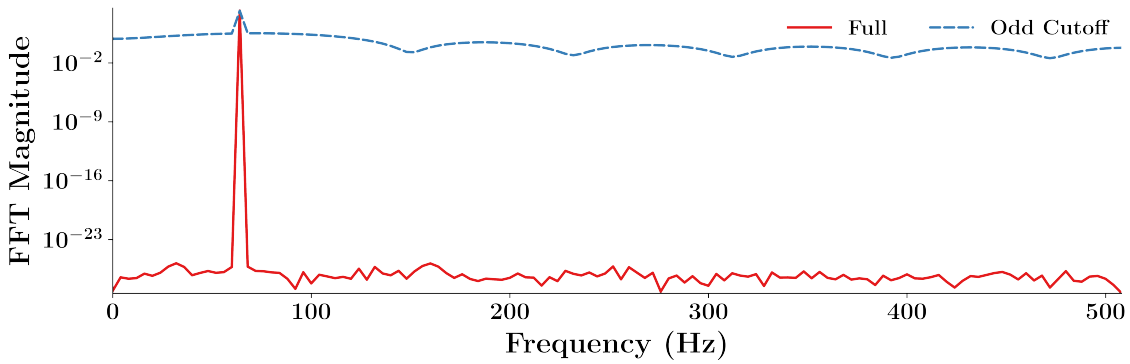


Figure 2.19: DFT of a signal with spectral leakage

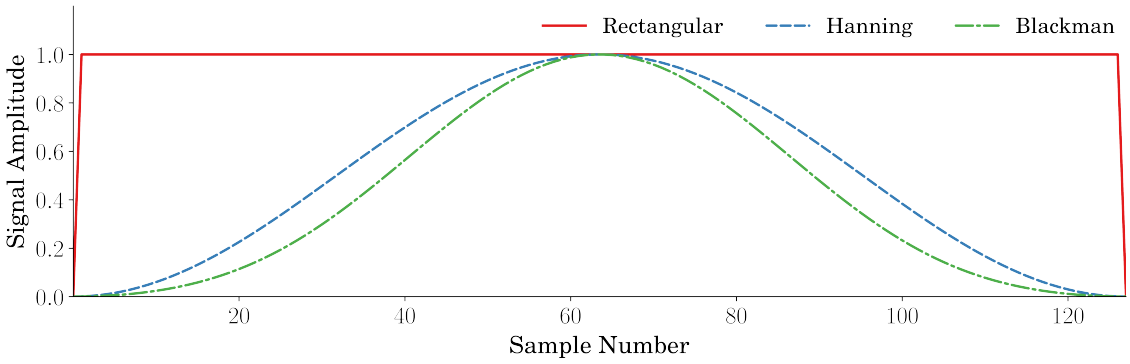
signal exhibits broadband noise at a level consistent with the minimum precision of floating point calculations, the Signal-to-Noise ratio of the signal with an odd cutoff is drastically reduced.

In practice, it is impossible to deterministically capture integer numbers of vibration signals. For this reason, measures must be taken to reduce the effect of spectral leakage. The most effective way of doing so is through a technique called *windowing*, in which the signal is multiplied by an aptly-named windowing function designed to mitigate broadband noise. A large variety of these functions have been developed, each with design trade-offs.

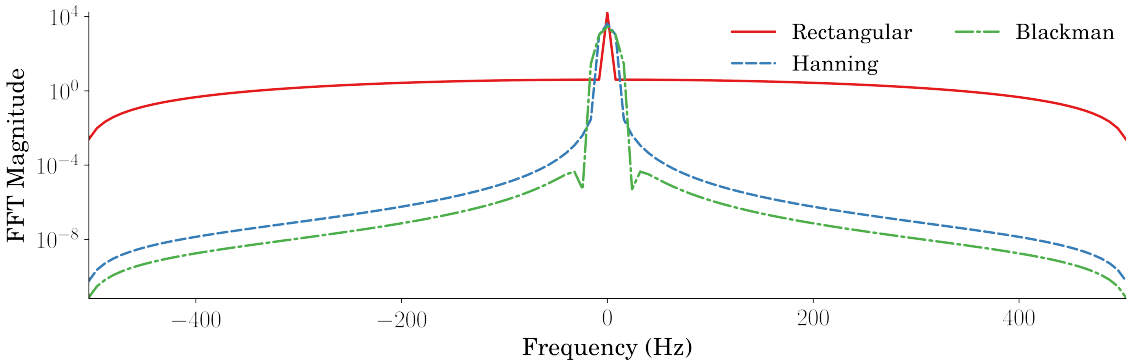
Figure 2.20 shows three different windowing functions to demonstrate their design strengths and weaknesses. The time-series plots of the Hanning, Rectangular and Blackman windows are shown in Figure 2.20a. The Rectangular window is clearly the most simple, as it has unity magnitude for the duration of the sample. However, there is a discontinuous step at the beginning and end, where the magnitude shifts from zero. Next, the Hanning window is defined by

$$w_{hanning}[n] = \sin^2\left(\frac{\pi n}{N}\right), \quad 0 \leq n \leq N, \tag{2.8}$$

and gradually approaches zero at the start and end of the sample. The slightly different



(a) Time series plot



(b) Fourier Transform

Figure 2.20: Comparison of windowing functions

Blackman window is defined by

$$w_{blackman}[n] = 0.42 - 0.5\cos\left(\frac{2\pi n}{N}\right) + 0.08\cos\left(\frac{4\pi n}{N}\right), \quad 0 \leq n \leq N. \tag{2.9}$$

Upon inspection in the time-domain, it is difficult to determine meaningful differences between these two windowing functions. However, their DFTs provide a clearer picture in Figure 2.20b. First, the abrupt steps at the beginning and end of the Rectangular window introduce high amplitude, broadband noise. This is mitigated by both the Hanning and Blackman windows due to their smoothing. Clearly, the Blackman window is slightly more effective at this noise reduction and appears to have a more distinct peak compared to the Hanning window. This benefit comes at the cost of a slightly wider peak than the Hanning window. While both of these smooth functions exhibit slightly lower amplitude at the

Table 2.1: Accelerometer Noise Density Comparison

Name	Price	Noise Density
ADXL 203	$19	$110 \frac{\mu g}{\sqrt{Hz}}$
PCB 352C04	$325	$4 \frac{\mu g}{\sqrt{Hz}}$

DC frequency bin, the overall signal-to-noise ratio is dramatically improved by reducing broadband noise.

2.5.4 Minimizing Noise

When analyzing a real signal, various sources of noise may obfuscate the underlying data. All sensors are subject to noise when capturing measurements. While sophisticated accelerometers exhibit low noise floors relative to their low-cost counterparts as shown in Table 2.1, they require equally sophisticated hardware and software to perform data acquisition. Beyond this, it is still beneficial to address and attempt to minimize noise introduced into vibration measurements.

While some factors such as variation in turning speed may have negative effects on signal processing and must be addressed by changing the data acquisition methods, some noise — especially sensor noise — can be minimized by breaking a signal into multiple segments, performing a DFT on each segment, and averaging the results. This is based on the assumption that sensor noise is Normally distributed and therefore has a mean of zero. By averaging multiple DFT measurements, the influence of such noise diminishes. Because the number of sample points is inversely proportional to DFT frequency resolution, this averaging reduces the effective sample duration and therefore the frequency resolution. However, by intelligently planning measurements to accommodate averaging, the quality of the sample can be improved.

2.5.5 Welch's Method

All of the previously mentioned methods for improving DFT quality can be combined to approximate the Power Spectral Density (PSD) of a signal. One widely used way of implementing these tools is known as Welch's Method [55]. In this method, a signal is broken into a specified number of subsections with a 50% overlap. These subsections are then multiplied by a windowing function of choice and their DFTs individually computed. Next, these DFTs are averaged together and their magnitudes squared to approximate the PSD. The result is an excellent approximation of the frequency content of a signal in the presence of sampling imperfections and sensor noise. Most modern signal processing software packages have tools for implementing this method with best-practice defaults and rules such as limiting the number of subsections for averaging.

2.6 Time-Domain Signal Processing

When analyzing time-series vibration data, meaningful statistical features must be chosen to properly detect damage progression. Statistical moments represent an important class of such features. The k^{th} raw moment of a discretely valued sample is given by

$$M^k = \frac{1}{n}\sum_{i=1}^{n} X_i^k, \tag{2.10}$$

where the first raw moment is simply the sample mean, $\bar{X}$. Similarly, the second central moment,

$$\sigma^2 = \frac{1}{(n-1)}\sum_{i=1}^{n}(X_i - \bar{X})^2, \tag{2.11}$$

yields the sample variance.

Increasing orders of moments are often represented in their standardized form. For

instance, the third standardized moment,

$$\text{Skew} = \frac{M^3}{\sigma^3}, \tag{2.12}$$

is known as skewness. This metric can be used to identify whether a signal favors one side of the distribution or another. Lastly, the fourth standardized moment,

$$\text{Kurtosis} = \frac{M^4}{\sigma^4}, \tag{2.13}$$

is called kurtosis. This measurement provides insight into the "tailedness" of a sample. A higher kurtosis value is indicative of a sample with a tendency towards more extreme values.

Of the statistical moments, kurtosis has been shown to be a robust indicator of bearing damage, although time-to-failure cannot be discerned from this statistic [56]. Later work combined kurtosis with Root Mean Square (RMS) in a Bayesian approach to model bearing degradation [57].

2.6.1 Time-Frequency-Domain Methods

Another subset of signal processing and analysis techniques involves the use of time-frequency metrics such as wavelets and spectral kurtosis. The use of the wavelet transform has been shown to be helpful in identifying defects in bearings [58, 59], gears [60], and gearboxes [61]. This transformation is defined by

$$W_\psi(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\psi^* \left(\frac{t-b}{a}\right) \mathrm{d}t, \tag{2.14}$$

where $\psi(t)$ is a basis wavelet, and a, b are dilation and translation factors, respectively. The higher dimensionality of the wavelet transform enables greater generalization compared to the Fourier Transform, including detecting impulsive transients present in defective

machinery. This comes at the cost of increased complexity required in its computation.

Spectral Kurtosis is another time-frequency metric which is useful in the categorization of vibration signals. This metric was formalized as a means of quantifying the "non-stationarity" of signals — that is, the transient qualities which may be indicative of bearing defects [52]. Because some bearing defects may be hidden by high frequency amplitude-modulated signals, it can be difficult to detect defect frequencies in the presence of this modulation. Using spectral kurtosis, it is possible to automatically perform enveloping to recover the true defect frequency [62].

2.7 Publicly Available Testing Datasets

When developing a methodology for recording and diagnosing machine health, it is useful to have a rigorously studied benchmark dataset which contains known failures. A number of such datasets have been made public for researchers to use [63, 64]. While these datasets can greatly facilitate the beginning states of prognostics research, it is important to note that their applicability is quite narrow, and substantial data acquisition efforts are required for more even slight deviations from these test conditions [65]. With that being established, significant meaningful insight can be gained by comparing the relative prognostic and diagnostic power of different machine monitoring algorithms and statistics with these benchmark data.

The IMS dataset [63], provided by NASA, is particularly useful for vibration analysis. This contains several run-to-failure experiments on bearings with multiple accelerometers continuously capturing data over the course of several days. For instance, the second trial in this collection contains four channels of accelerometer data captured at ten minute intervals over seven days. Each channel corresponds to the data from an accelerometer mounted on one of four bearings. Towards the end of this trial, it is noted that outer race failure occurred in bearing 1.

The RMS values of the vibration data from this trial are shown in Figure 2.21. Clearly,

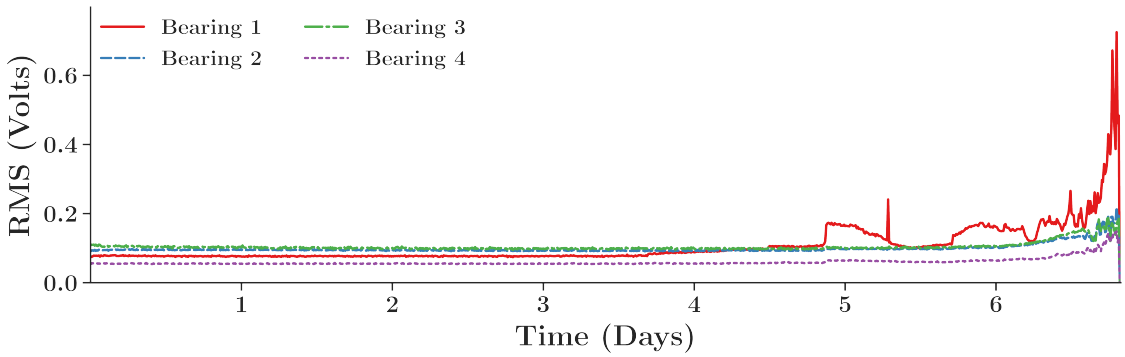
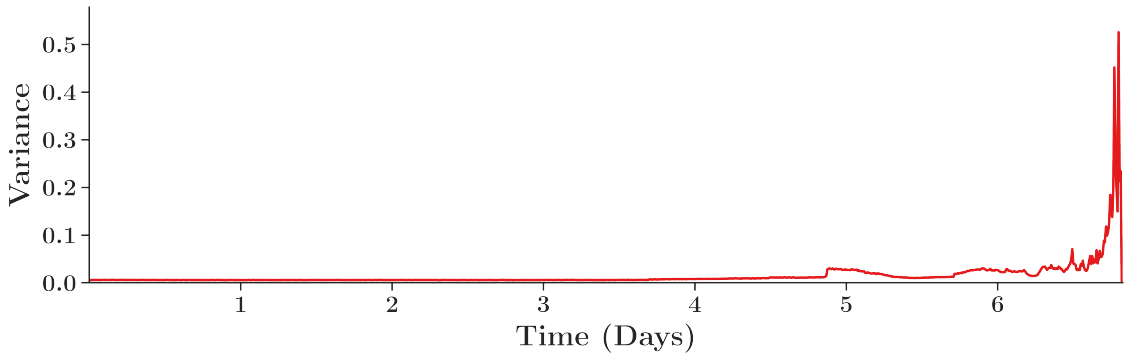


Figure 2.21: Root Mean Square history of IMS test dataset

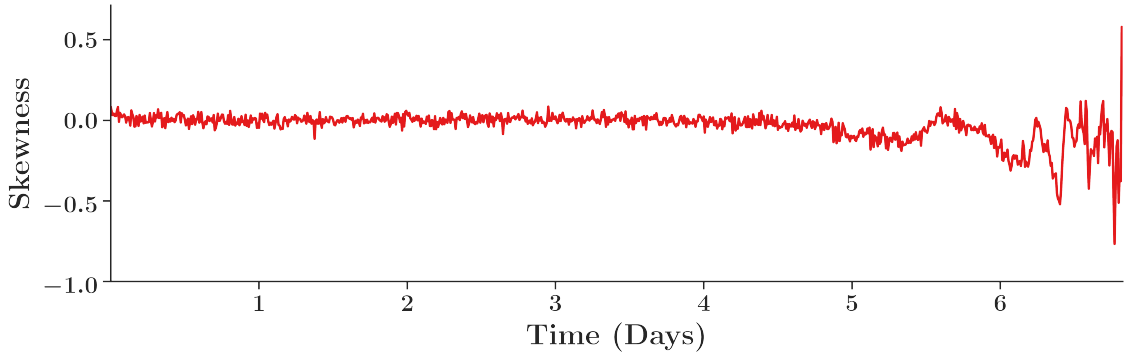
the intensity of vibration changed around Day 5 in Bearing 1, which seems to be an early indicator that a defect has manifested. By comparison, the vibration sensors on the other three bearings show minimal changes in their RMS values. Only toward the very end of the test, when the vibration signature from Bearing 1 has dramatically increased in magnitude, do the other signals begin to change.

To illustrate the potential usefulness of the statistical moments discussed in Section 2.6, the variance, skewness, and kurtosis history of Bearing 1 are shown in Figure 2.22. Figure 2.22a shows the variance history. While the bearing is healthy, the variance of its vibration signal is near zero. Late in Day 4, the variance jumps as early damage manifests. After this point, the variance drops again before gradually increasing until late in Day 6 where failure occurs. This trend is generally repeated in Figure 2.22b, which shows the skewness history. The healthy bearing has near zero skewness, indicating a normal distribution. Near the familiar times at Day 5, a change is evident as the vibration measurements develop a negative skew. As damage advances, the skewness of the signal becomes increasingly erratic until failure. Finally, the kurtosis is shown in Figure 2.22c. This signal follows a very similar trend to the variance. Because kurtosis is interpreted as the tendency of a signal towards more extreme values, it follows that kurtosis should increase while variance and skewness also increase.

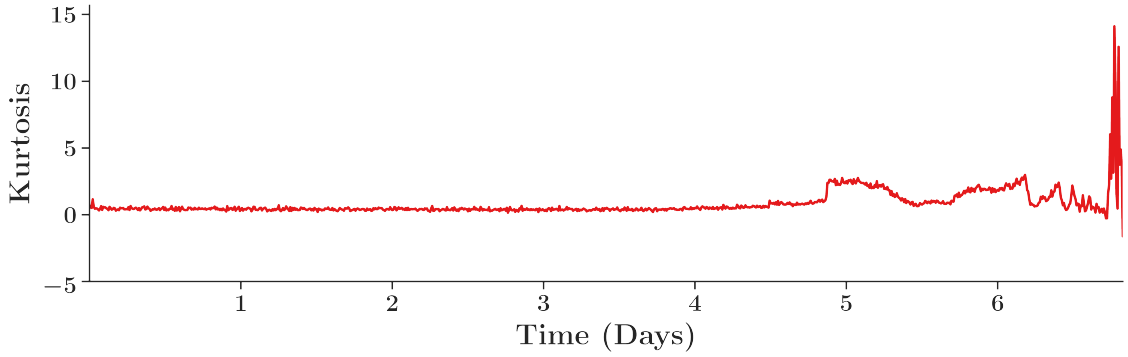
A similar trend can be found in the spectrogram of the vibration data from Bearing 1,



(a) Variance history



(b) Skewness history



(c) Kurtosis history

Figure 2.22: Statistical moments for Bearing 1 in the IMS dataset 2

shown in Figure 2.23. As is clear from this plot, the frequency content of this signal is static for about four days. At the beginning of day four, small, but distinct, changes are evident in the spectrogram. These changes indicate the potential presence of defect frequencies in the signal. By day five, the high frequency signals begin to manifest more clearly. This trend continues until the end of the experiment. Note here that the changes in the spectrogram

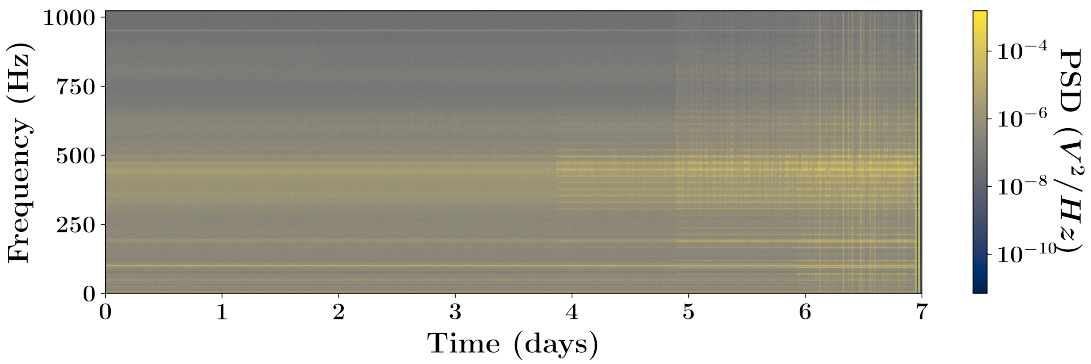


Figure 2.23: Spectrogram of Bearing 1

can be easily identified a full day before changes in the time-domain statistical features are apparent.

CHAPTER 3

A DIGITAL ARCHITECTURE FOR MACHINE HEALTH MONITORING

This chapter describes the methodology and design of an architecture for transmitting machine data from a factory floor to cloud storage for analysis and historical monitoring. Specifically, this architecture leverages modern web protocols such as MQTT to enable low-overhead bidirectional communication for data acquisition devices in a factory setting. Using a standardized messaging format, communication across a variety of machines may be facilitated. This architecture also incorporates the use of additional sensors for health monitoring. By leveraging CNC controller data to indicate the current operational state of a machine, sensor data are readily contextualized relative to machine status. Two simple case studies illustrate the usefulness of sensor data in detecting tool wear and tracking spindle health in a milling machine.

3.1 Background

A typical IoT-connected Smart Factory is shown in Figure 3.1. Various factory equipment and sensors must be networked to a gateway capable of securely routing machine data to an industrial network. From the industrial network, machine data must then be routed to cloud computing and storage resources. These data are then made available to an end user tasked with tracking the health and efficiency of the the production floor. This architecture is commonly used in modern IoT systems [66].

When considering this Smart Factory architecture, several factors affect the ease with which machine data are made available to a variety of monitoring applications. First, it is important to standardize the data format from sensors and machines on the factory floor. This can be challenging. Various manufacturing equipment suppliers utilize an array of CNC controllers and protocols for accessing machine data. Beyond this, specific data tags

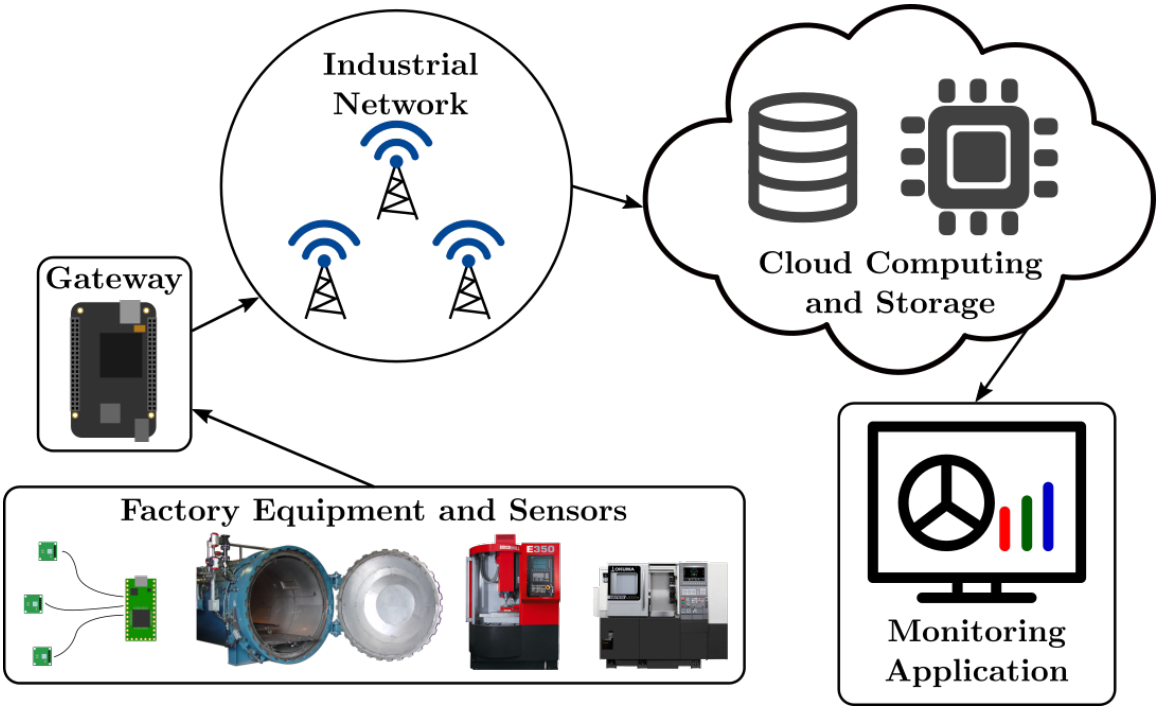


Figure 3.1: Typical smart factory

may have different names from machine to machine, even from the same manufacturer. For instance, the spindle speed on one CNC lathe may be identified by the tag "LS1Speed" and "S1Speed" on another. Next, these data must be efficiently routed through the industrial network in a way that makes them readily available for backend storage and applications. To accomplish this task, machine data should be standardized prior to transmission. It is also important to create a standardized method for storing these data for later use.

3.2 Proposed Architecture

The proposed architecture is shown in Figure 3.2. The bottom of the image shows the various machines and communications protocols which may be integrated into this framework. Various machine vendors support different protocols such as MTConnect or OPC-UA for their equipment. To ensure a common data structure which accommodates these CNC protocols and external sensor data, a JSON-based messaging format is used. Gateways are connected to the manufacturing equipment in order to allow data to flow

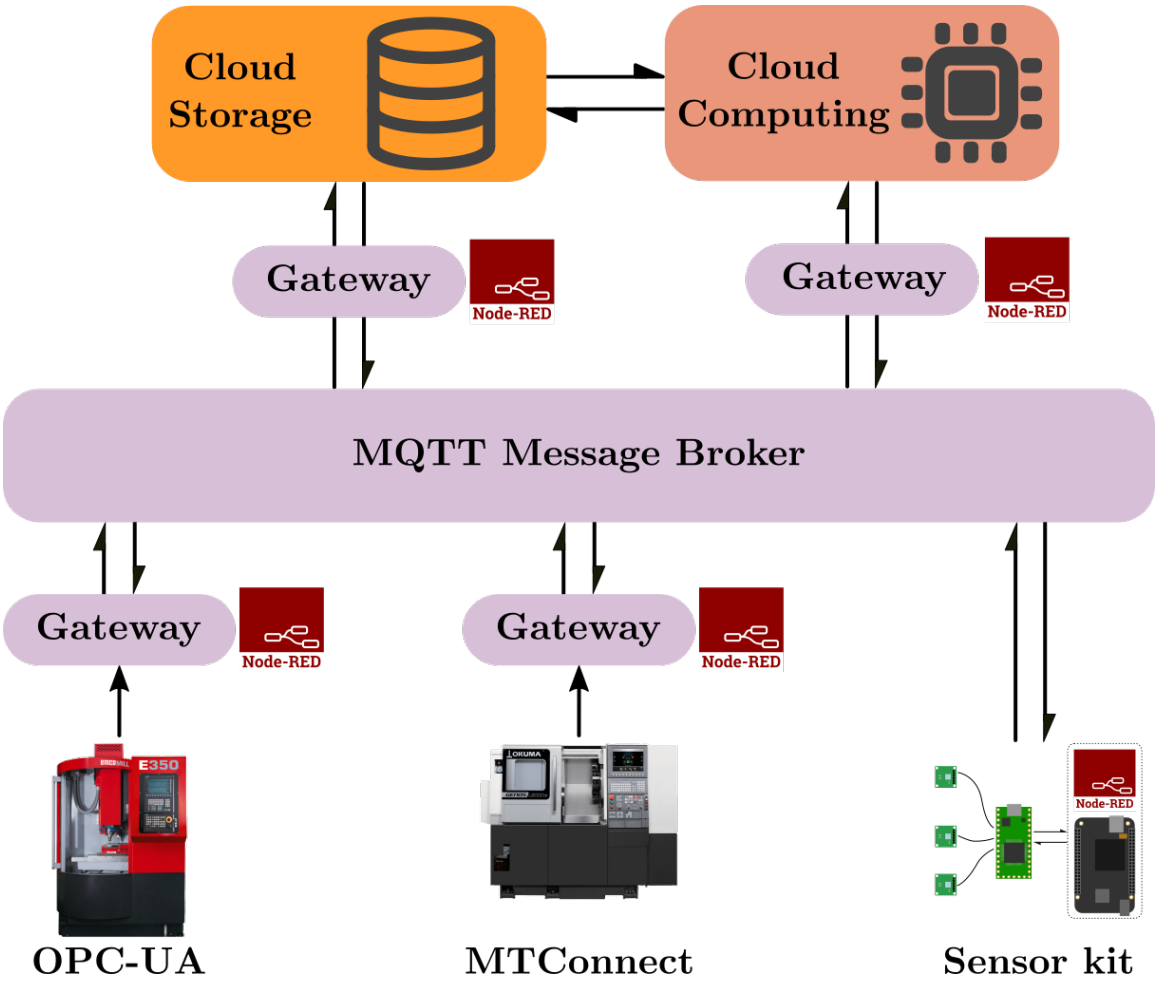


Figure 3.2: Proposed digital architecture

from the machines to the enterprise network. For the purposes of this work, a Gateway may be considered as any general purpose computer capable of interfacing with the machine controller and publishing to the MQTT message broker. To provide flexibility and accommodate the various data sources, each Gateway in this work uses Node Red to receive, parse, and publish machine and sensor data. Upon exiting these gateways, the transmitted data are in a standardized JSON format which allows for multiple different machines to be used within the same framework. An MQTT message broker serves as an intermediary for all data, which can then be routed to the desired endpoint. For example, historical data can be stored in databases and real-time utilization information can be pushed directly to web-based applications. The modularity of this approach permits

scalability; to accommodate more machines, one must simply increase the computational power of the MQTT message broker.

It is worth noting that nearly all of the connections in this architecture are bidirectional. The publish-subscribe nature of MQTT allows for machines to transmit and receive data. Importantly, the OPC-UA and MTConnect connections to the gateways do not follow this pattern. In the case of MTConnect, this is by definition a read-only protocol. In contrast, OPC-UA allows for read-write permissions, which can be a security and safety issue. Because it is not necessary to remotely write data to a CNC machine for the purposes of health monitoring, the connection between the OPC-UA server and its gateway is considered as read-only in this work.

Also note that this architecture accommodates a sensor kit with no external Node Red gateway. The sensor kit is a self-contained IoT device capable of performing data acquisition and formatting that data in a manner that is readily consumed by the MQTT message broker. This fact can be exploited to remotely update and trigger data acquisition, whether by predefining MQTT-based rules or by reprogramming the Node Red interface. For example, if a sensor kit is capturing data from an MTConnect-compatible machine, the sensor kit may subscribe to specific data items from the CNC machine and only perform data acquisition when these data items meet a certain criteria.

3.3 Message Format

A normalized messaging format enables standardized data transmission across the proposed architecture. As part of this messaging format, a defined MQTT topic structure facilitates client access to desired data streams. In addition, standardized payload fields and definitions are used to ensure easy message parsing throughout the architecture. This topic and payload format are presented in the following sections.

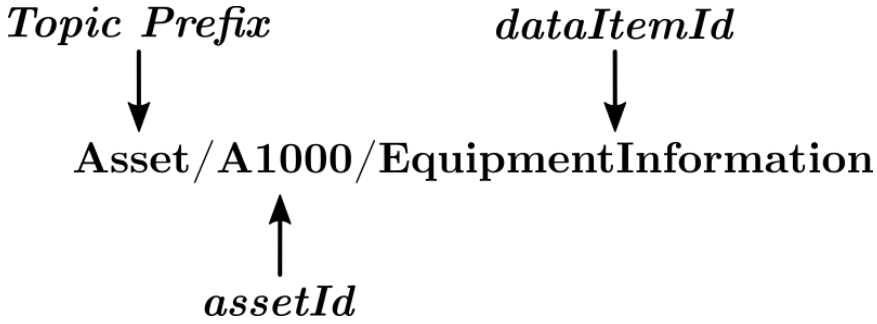


Figure 3.3: MQTT topic structure

3.3.1 Topic Structure

A proposed topic structure for this work is shown in Figure 3.3. This structure contains three levels. First, the "Asset" tag is used to indicate the type of topic. This tag indicates that the next level will contain an *assetId*. Although it may seem redundant, it allows for the same MQTT topic structure to be used for other types of messages beyond what is discussed in this thesis. The *assetId* field is a unique identifier for a factory asset such as a CNC machine. The next level is the *dataItemId* of the message payload. This could refer to a specific data tag from the CNC controller or a sensor reading such as vibration, temperature, or current.

Within this context, it is possible that multiple IoT devices transmit data associated with the same *assetId*. For instance, a sensor device may be mounted on the spindle of a CNC mill which is also transmitting controller data. Both the data from the sensor and CNC controller have the same *assetId* in their respective MQTT topics. This guideline ensures that IoT devices which are tracking different *dataItemIds* from the same machine may be easily configured to listen to one another. To do so, they must simply be subscribed to the appropriate *assetId*.

3.3.2 Payload Structure

With the topic structure defined, it is possible for any device to connect to the MQTT message broker and subscribe to a desired data stream. Next, a payload structure must also

Table 3.1: JSON message definitions for basic controller data

Field	Description
assetId	An identifier for the machine or device
dataItemId	An identifier for the type of data
dateTime	The time at which the value was taken
value	The numerical or string value of the data being sent
itemInstanceId	An optional identifier for the item being processed

```
{
"topic": "Asset/A1000/rotaryVelocity_S",
"payload": {
    "dateTime": "2019-03-08T15:43:44.808Z",
    "assetId" : "A1000",
    "dataItemId" : "rotaryVelocity_S",
    "value" : 1002
    }
}
```

Figure 3.4: Example controller payload

be defined so any connected application can readily access desired data fields. The baseline fields for the proposed JSON payload are summarized in Table 3.1. The first two fields in this table are simply repeated from the MQTT topic. Next, the *dateTime* field is an ISO string containing the time at which the *value* contained in the payload was captured. This time is recorded to the millisecond, which is the maximum precision of JavaScript. The *value* field contains the string or number associated with the *dataItemId* in this payload. The data type depends on the *dataItemId*. For example, if the *dataItemId* relates to a G Code block, the *value* will be a string because G Code is, by definition, a string.

An example of the standard MQTT format is shown in Figure 3.4. The whole MQTT object is a JSON string with a *topic* and *payload*. In this example, the *assetId* is "A1000", and the *dataItemId* is "rotaryVelocity_S", indicating the rotation speed of the spindle. The *value* field contains an integer speed in revolutions per minute, as read from the controller. While this payload structure may cover a broad range of basic *dataItemIds*,

```
{
"topic": "Asset/A1000/EquipmentInformation",
"payload": {
    "dateTime": "2019-03-08T15:43:44.808Z",
    "assetId" : "A1000",
    "dataItemId" : "EquipmentInformation",
    "informationId" : "ProgramName",
    "value" : "WARMUP"
    }
}
```

Figure 3.5: Example equipment information payload

it is necessary to specify a standard for other data types and categories of machine information. A full list of the various *dataItemIds* and their associated message structures is maintained online [67]. One specific payload structure which will be important in this work is the `EquipmentInformation` *dataItemId*. This data item is used to transmit operational and health information pertaining to the asset. An example message is shown in Figure 3.5. This figure shows the additional field, *informationId*, which is used to assign the specific category of the equipment information being transmitted. In this example, the `ProgramName` identifier indicates that this payload contains the current name of the program being executed by the piece of equipment.

Beyond equipment and controller information, this architecture is designed to transmit sensor data from IoT sensor packs. Sensors such as accelerometers, current sensors, thermocouples, microphones, and cameras may transmit data via MQTT, and their various structures are also documented online [67]. Because this thesis focuses on vibration analysis, the proposed message structure for vibration data is shown in Figure 3.6. This message format is intended to be used in health monitoring applications, and contains all of the summary statistical moments and frequency information from an accelerometer, as discussed in Chapter 2. The `samplingInterval` indicates the time between vibration samples in seconds. The first four statistical moments — mean, variance, skewness, and

```
"topic": "Asset/A1000/Vibration",
"payload": {
    "dateTime": "2019-03-08T15:43:44.808Z",
    "assetId" : "A1000",
    "dataItemId" : "Vibration",
    "iteminstanceId" : "Part_1001",
    "sensorId": "accelerometer1",
    "samplingInterval":0.00048828125,
    "Kurtosis": 0.03314028430295357,
    "mean": -0.040322119999999996,
    "rootMeanSquare": 0.0248660134381368,
    "skewness": -0.20419674499921678,
    "variance": 0.0006189375618674674,
    "PSDFreqInterval":10,
    "PSDAmps":[0.895,0.889,...,0.877]}
}
```


Figure 3.6: Example vibration payload

kurtosis — are provided as well. Root Mean Square (RMS) is also provided as an additional parameter which may be of interest in health monitoring. Lastly, the PSD information is provided by the final two fields. The first field, `PSDFreqInterval`, provides the difference between frequency points on the Fourier transform in Hertz. The second, `PSDAmps`, contains an array of the PSD magnitudes.

3.4 Machine Health Monitoring Application

Machine and process health monitoring is a complex area of study in and of itself, and it is important to establish a procedure for utilizing the proposed architecture to facilitate these tasks. An interpretation of machine health monitoring is shown in Figure 3.7. In this figure, a variety of machines generate large quantities of data. For instance, CNC machines may transmit positional axis information and motor loads. These data streams may be enriched by sensors such as accelerometers, which produce extremely dense data samples and frequency-domain information. Within this application, all of these data sources are

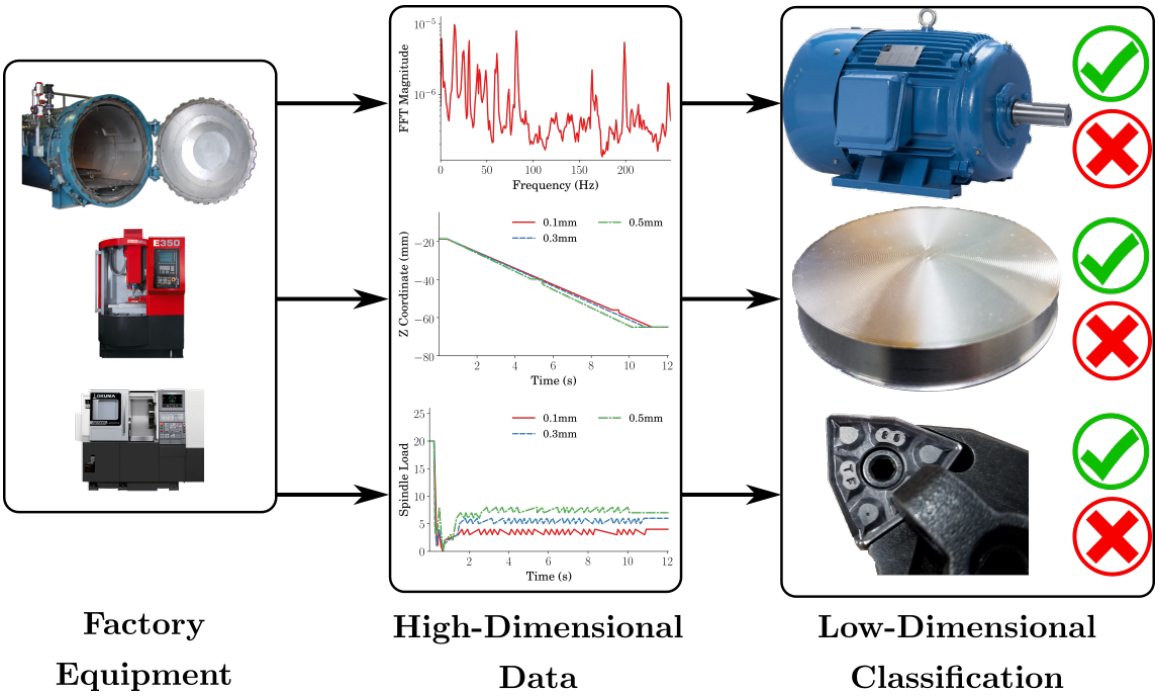


Figure 3.7: Architecture application in health monitoring

"high-dimensional," that is, one stream of data from a particular manufacturing operation on a particular machine may have many points. When determining a classification such as motor health, part quality, or tool wear, these data streams are typically condensed into substantially lower dimensionality, such as a binary "good/bad" categorization or a probability that the data are healthy/unhealthy.

Within the context of the Internet of Things, and considering that a single factory may contain hundreds of unique machines and potentially thousands of sensors, it is important to create an architecture which intelligently uses limited resources such as network bandwidth and data storage to facilitate the necessary classifications. Continuously transmitting raw controller and sensor data is not an efficient use of these resources. Such an approach forces network and cloud computing resources to constantly perform the high-dimensional to low-dimensional conversion. Furthermore, if controller and sensor data are used to help track the quality of a manufactured part, it is important to capture and store this data relative to the context of the machining operation.

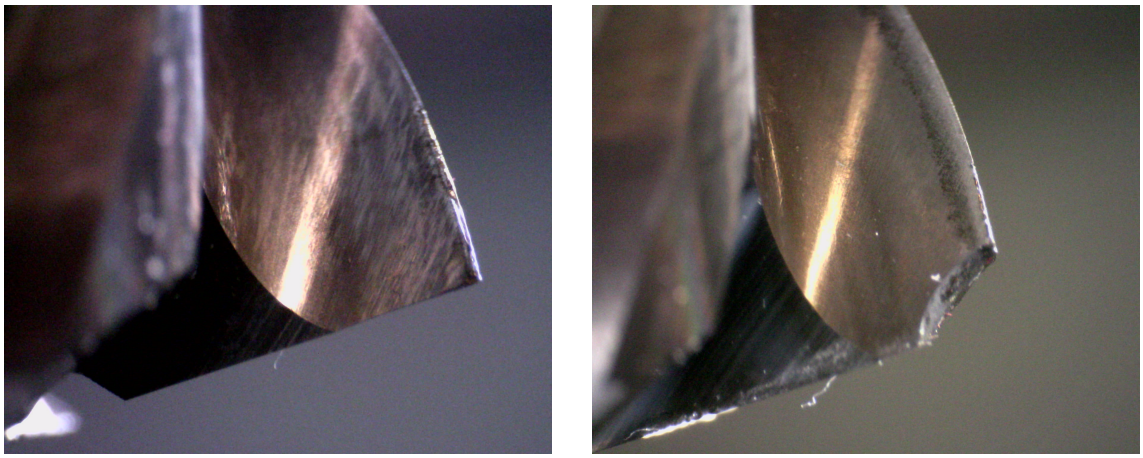
3.4.1 On the Utility of Controller Data

While modern CNC controllers provide vast amounts of machine data, a number of issues limit the utility of these data for machine health monitoring. First, because various manufacturers use different nomenclatures for their controller data, these data tags must be manually sorted through to identify those that are potentially useful. Once selected, it may take some trial-and-error to determine which of the down-selected tags are actually helpful for health monitoring. Second, the controller data often have undocumented sampling and data formatting features. Details such as sampling rate, anti-aliasing filters, and quantization are often left to the operator to determine. These issues reduce the potential utility of controller data in health monitoring applications.

To help illustrate these downsides, consider the controller data from the Emco E-350 3-Axis mill. Because spindle power consumption is a good measure of machine health and utilization, this is a useful data item to capture. To do so, a technician must interface with and sort through the OPC-UA controller data. For this machine, OPC-UA server contains this value at the $/MachineAxis/aaPower$ address.

To evaluate the usefulness of this metric, a series of cuts are performed. Two four-flute end mills — an unused, 'nominal' cutter and a heavily worn, 'anomalous' cutter — are used to cut slots in 6061 aluminum. Close-up images of the tool cutting edges are shown in Figure 3.8. The nominal cutter is shown in Figure 3.8a. The leading edge of the cutter is sharp, with no signs of wear. By comparison, the anomalous tool, shown in Figure 3.8b, has an extremely large chip in the leading edge. These images are typical of each of the four flutes for the respective tools. It is worth noting that the wear visible in the anomalous tool is sufficiently advanced such that it would be considered unusable in a production environment. As such, this advanced level of wear must be easily visible from the machine monitoring data.

An accelerometer is mounted to the Emco spindle for comparison with the controller data. The workpiece setup is shown in Figure 3.9. According to prior work [68, 69, 70],



(a) Nominal tool (b) Anomalous Tool

Figure 3.8: Tool quality comparison

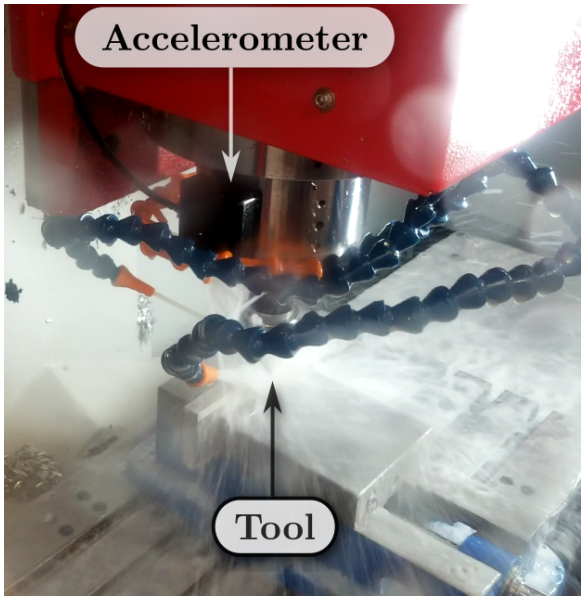


Figure 3.9: Emco Mill setup

the heavily worn tool should exhibit greater vibrations and require more power to cut the material due to increased rubbing from the dull teeth. The power consumption of the spindle for the nominal and anomalous case is shown in Figure 3.10. While it is apparent that the heavily worn tool causes an increase in the power consumption as reported by the CNC controller, the values have high variance and are insufficiently separated to avoid Type I and Type II error based on the controller data alone. Again, due to uncertainties in the

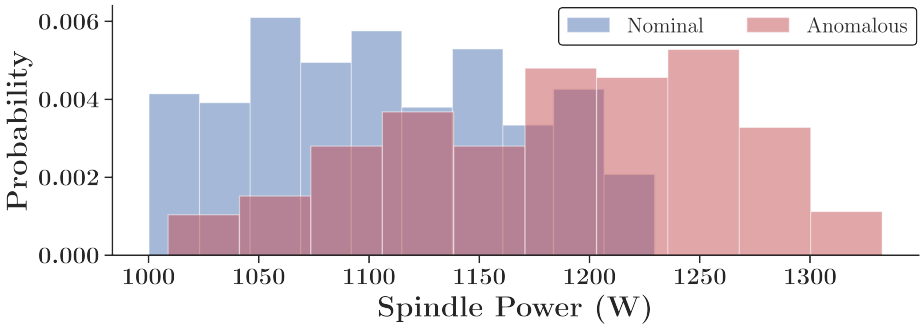


Figure 3.10: Controller spindle power comparison

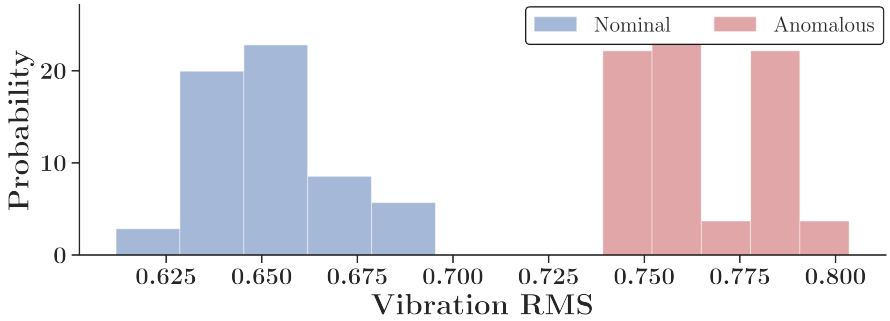


Figure 3.11: Accelerometer RMS comparison

sampling and filtering methodology from the controller, it is difficult to determine a best course of action to improve the separation between these two datasets. Also, because the anomalous tool is in such an advanced state of wear, it is troubling that the controller data are so closely grouped.

In contrast to the controller data, the accelerometer data reveal a significant separation between the worn and unworn tool. This is illustrated in Figure 3.11, which shows the RMS vibration values captured during both cutting operations. These two groups are distinct, with no overlap. While a low-dimensional metric such as RMS or spindle power may be convenient to track the health of a tool, more high-dimensional tools such as the Power Spectral Density can be significantly more informative and offer more nuance to the health monitoring algorithm. Such a comparison is shown in Figure 3.12. Because

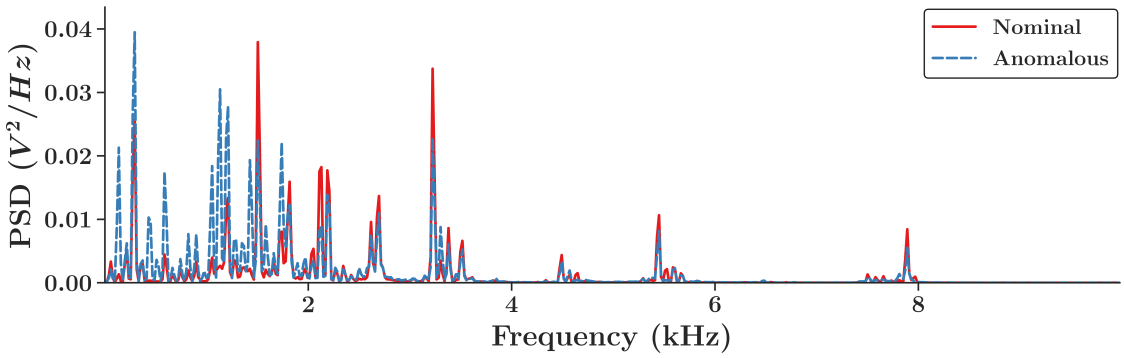


Figure 3.12: PSD Comparison for nominal and anomalous tool

the PSD contains a large number of dimensions corresponding to the different frequencies in a signal, it is easy to identify individual components which are more prevalent in the anomalous signal. This increase in dimensionality allows for more in-depth information regarding the specific mechanisms which differentiate the nominal and anomalous tool.

While this is a highly simplified example application of health monitoring, it illustrates the superiority of specialized sensors in capturing nuances in health metrics when compared to CNC controller data. For these reasons, the machine controller data are used exclusively for process contextualization, while external sensor data are used for health classification in this framework.

3.4.2 Sensor Data Contextualization

Using the proposed architecture, it is possible to capture, store, and analyze data from various sources with the objective of creating intelligent, enterprise-level health monitoring systems. For example, sensor data may be pushed into a database table while machine operational information is pushed into a separate table. This process is summarized in Figure 3.13. Because both data streams contain *assetId* and *dateTime* information, it is relatively straightforward to use the EquipmentInformation table to dictate which vibration payloads are relevant for training a particular model. This process is demonstrated in Figure 3.14. In this example, a machine runs a warmup program where the spindle

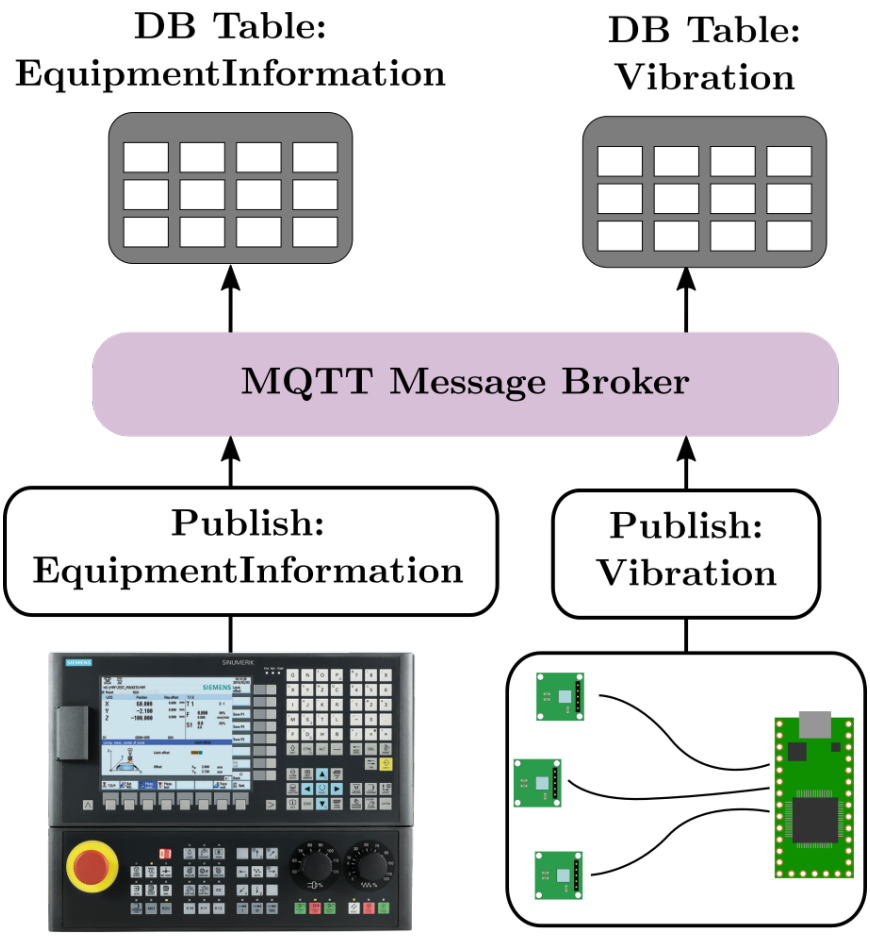


Figure 3.13: Contextual data acquisition

spins at a specified speed for several minutes. The CNC controller transmits payloads indicating the name of the current program and the commanded spindle speed. In parallel, an accelerometer transmits vibration data. To train a model capable of detecting changes in the vibration history of the spindle during this warmup program, the timestamps from the EquipmentInformation table are used to bound the relevant vibration payloads.

This architecture also facilitates "closing-the-loop" on statistical modeling and model fitting, as illustrated in Figure 3.15. The MQTT message broker allows factory machine controllers and external sensors to effectively exchange utilization and health information. While the controllers contain detailed contextual information such as job number identifiers, executed program names, and current operational states, external sensors lack this context. However, a sensor such as an accelerometer may extract vital

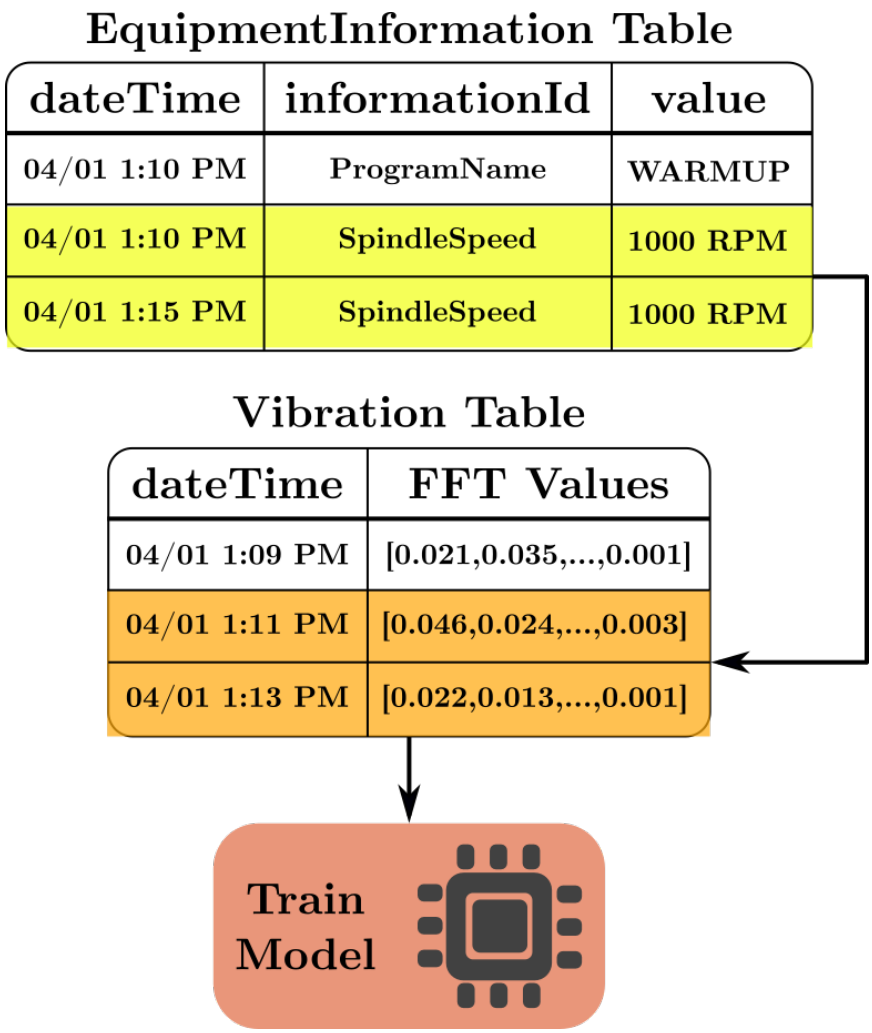


Figure 3.14: Training example from controller and vibration data

health information from a piece of equipment which cannot be captured with detail and precision by a machine controller.

By effectively labeling the vibration data with the equipment operational information, statistical and machine learning models may be derived from the sensor data through cloud computing. These models may then be executed at an arbitrary point in the architecture to track the health and utilization of the machine over time. For instance, edge computing may be leveraged to classify the health of the machine from the IoT sensor kit. Before this can be done, all sensor data must be transmitted through the architecture so that a baseline can be established. The architecture data throughput required for this activity is illustrated

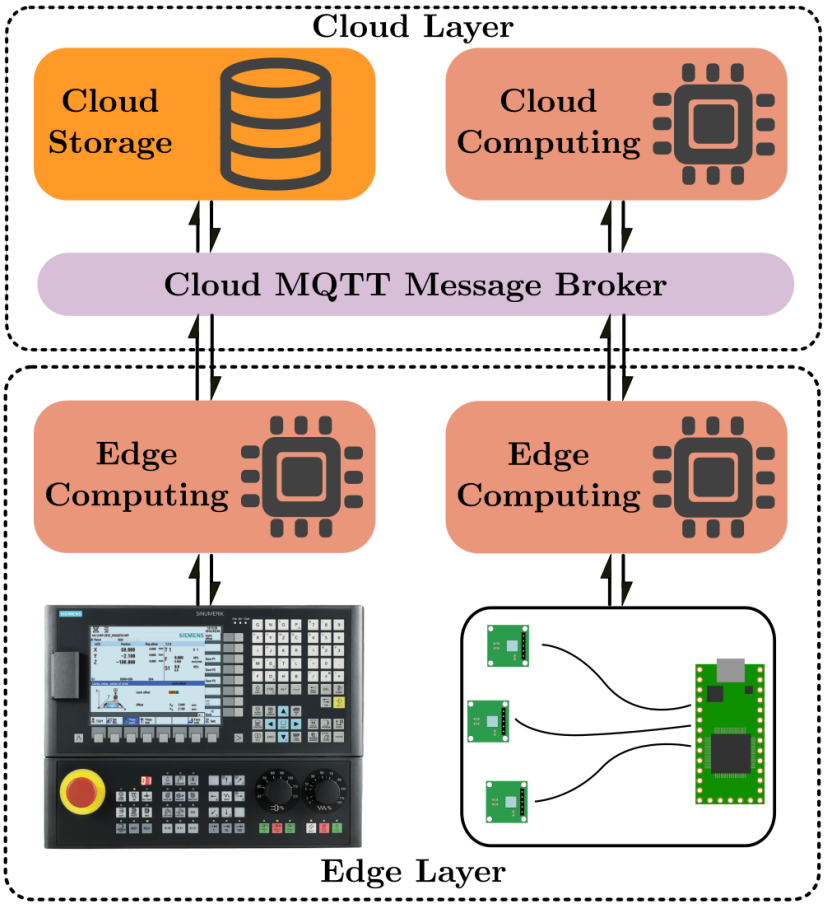


Figure 3.15: Computational resource distribution in the proposed architecture

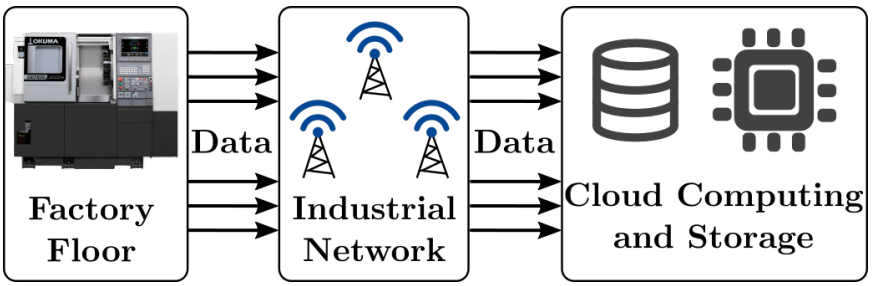


Figure 3.16: High data throughput from factory floor to cloud storage

in Figure 3.16. During this time, a large amount of data must be transmitted and stored so that statistical models can be built. After a period of time, data analysts and engineers may build statistical models to establish and track the health of factory machines over time.

With a statistical model, it is unnecessary to transmit the entirety of the sensor data

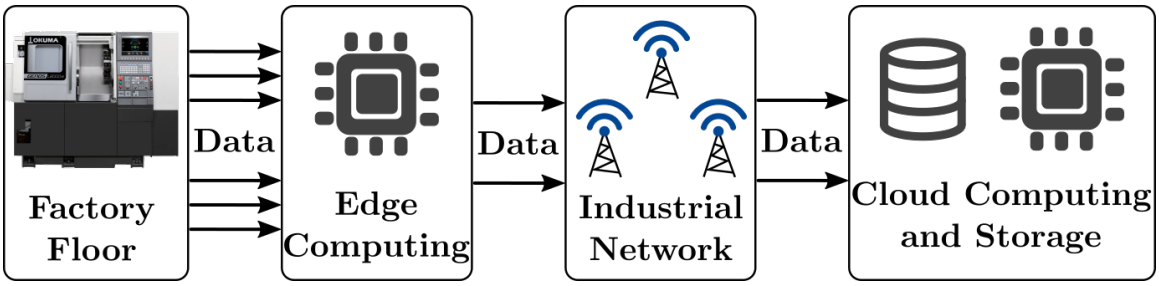


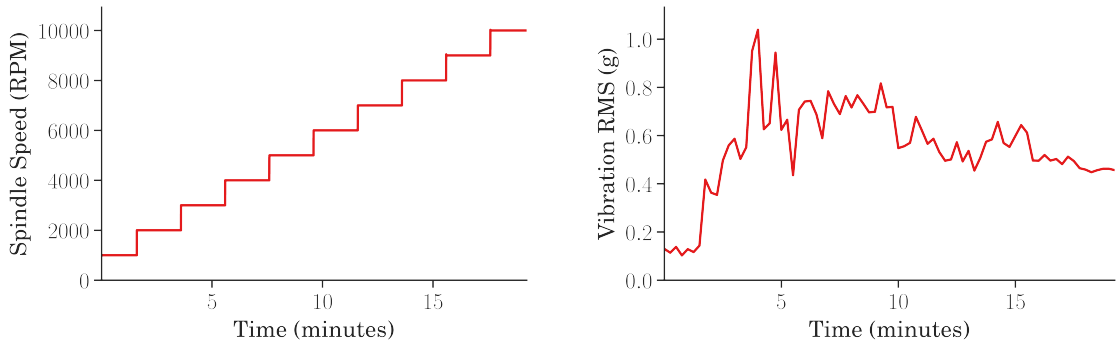
Figure 3.17: Reduced data throughput from factory floor with edge computing

through the architecture. Instead, edge computing can be used to compare new data to the baseline model and transmit a measure of how closely those data fit to the nominal case. This use of edge computing is illustrated in Figure 3.17.

3.5 A Case Study in Spindle Vibration Monitoring

To prevent damage to milling machines, they must be warmed up prior to operation. This is typically done by way of a simple CNC program which incrementally spins the spindle at higher speeds for several minutes. Because of the repetitive, consistent nature of this warm-up routine, it represents an ideal application for the proposed digital architecture for health monitoring.

Using the same Emco E-350 3-axis mill as in the previous section, data from the warm-up program were collected using the proposed digital architecture. The summary data typical of this program are shown in Figure 3.18. Every morning on a day in which this machine is used, the spindle is ramped up from 1000 to 10000 RPM, as Figure 3.18a illustrates. This program increases the spindle speed in increments of 1000 RPM, pausing for two minutes at each speed. This long dwell time provides an excellent window for steady-state vibration measurements at each spindle speed. The RMS vibration measurements from this program are shown in Figure 3.18b. Each RMS value corresponds to one second of vibration data. Because vibration measurements are taken at 15-second intervals, approximately eight measurements are taken at each spindle speed.



(a) Spindle speed

(b) Vibration RMS

Figure 3.18: Emco warm-up program typical data

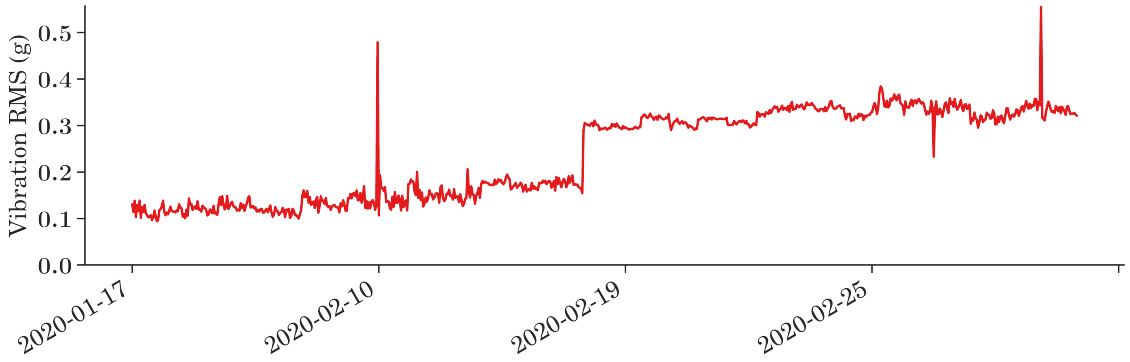


Figure 3.19: RMS history for warm-up program

Over the course of approximately one month, data from the warm-up program on this machine were captured. The RMS history of the vibration measurements at the 1,000 RPM speed are shown in Figure 3.19. On February 17th, the operator noted unusual behavior in the machine, specifically at the 1,000 RPM level. The machine began making an unusual noise and vibrating at this speed. In capturing the daily vibration levels leading up to this event, it is easy to see the point at which the vibration signature changed. Because the machine performance was not negatively impacted, regular use continued. While the odd behavior also continued, it did not worsen.

3.6 Conclusion

This chapter has presented and demonstrated the use of a digital architecture and messaging framework for machine health monitoring. The standardized JSON message structure facilitates scalability, data storage, and retrieval. The trade-offs between CNC machine controller data and external sensor data were also investigated. To ensure data accessibility and process contextualization, both data sources are required. Within the proposed framework, both data sources are shown to be useful in understanding the health and operational status of a machine.

With large quantities of high-quality sensor data captured from a CNC machine and labeled by the controller data, it is possible to create machine learning and statistical tools to assess machine health status. The next chapter will address modern tools and methods which may be used within this framework. Specifically, edge-deployable tools will be presented for the purpose of near real-time health monitoring on the edge.

CHAPTER 4

EDGE-DEPLOYABLE STATISTICAL ANALYTICS TOOLS

This chapter contains a discussion on the various machine learning techniques and models which may be applied in an IoT architecture for health monitoring. Leveraging advances in embedded computing and open-source statistical software libraries, advanced analysis and model inference are performed in low-power devices. A number of statistical and machine learning methods are presented for health monitoring applications. These methods generally fall under one of two categories: supervised and unsupervised learning. This chapter explores some of the more common methods and models used in both instances. Furthermore, a case study with simulated frequency-domain data is used to provide concrete examples of these methods. A variety of models are generated using open-source software and evaluated for their complexity, ability to generalize beyond the training data, and ability to be implemented in an IoT deployment. Using state-of-the-art open source machine learning tools, neural networks are deployed to embedded Linux devices with minimal performance loss. This finding supports the use of edge devices for near-real-time statistical inference on sensor data.

4.1 Overview of Statistical and Machine Learning Tools

Over the past decade, statistical and machine learning software has seen exponential growth in capability and popularity. Facilitated by the Python language [71], powerful statistical analysis and prediction algorithms have been packaged for use by the scientific community. In support of these tools, a wealth of knowledge has been placed in tutorial textbooks on their implementation [72, 73, 74]. Scikit-Learn is one such example [75]. Released in 2010, this software package has enjoyed mainstream adoption by the scientific community as indicated, by its Google Scholar citations, shown in Figure 4.1. Central to

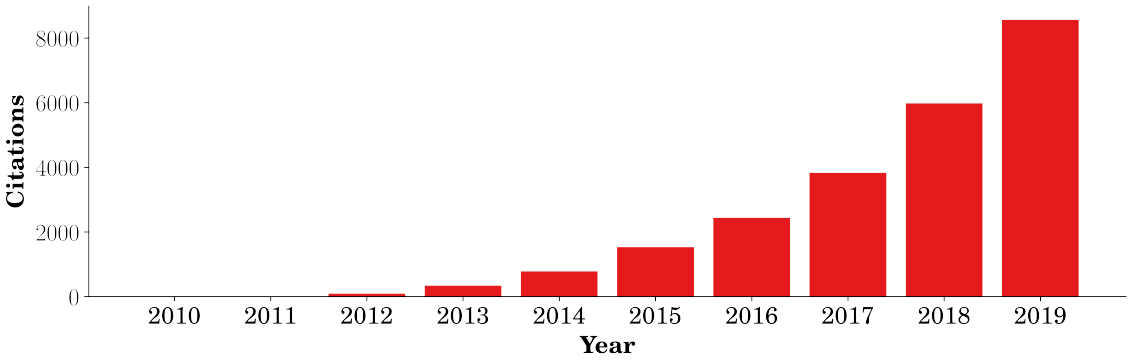


Figure 4.1: Number of citations for the Scikit-Learn Python library

the success of Scikit-Learn is a simple, consistent framework for building and assessing the performance of a large variety of machine learning algorithms. Built upon the powerful SciPy and NumPy [76] libraries, Scikit-Learn is designed to be deployable to any device which can accommodate these tools, including embedded Linux computers. In addition, Scikit-Learn and its basis libraries are distributed under the permissive BSD 3-clause license. By licensing these software packages in this way, machine learning algorithms may be developed and integrated into patentable intellectual property at no cost.

In the realm of deep learning and neural networks, TensorFlow [77] has similarly experienced widespread adoption in the past five years as Figure 4.2 illustrates. With the backing of Google, TensorFlow also enjoys mainstream adoption in a variety of industries. This widespread use is again facilitated by permissive licensing and open-source code. Similarly to Scikit-Learn, TensorFlow is distributed under the Apache 2.0 license. Again, this license allows for private, patentable use of derivative works as long as the original copyright and warranty disclaimers are kept intact.

While deep neural networks are computationally expensive to train and typically require a dedicated graphics card to do so, TensorFlow is capable of generating optimized models which can be readily deployed to embedded computers and even microcontrollers for inference. This has led to an increase in research interest for developing small, embedded machine learning algorithms [74].

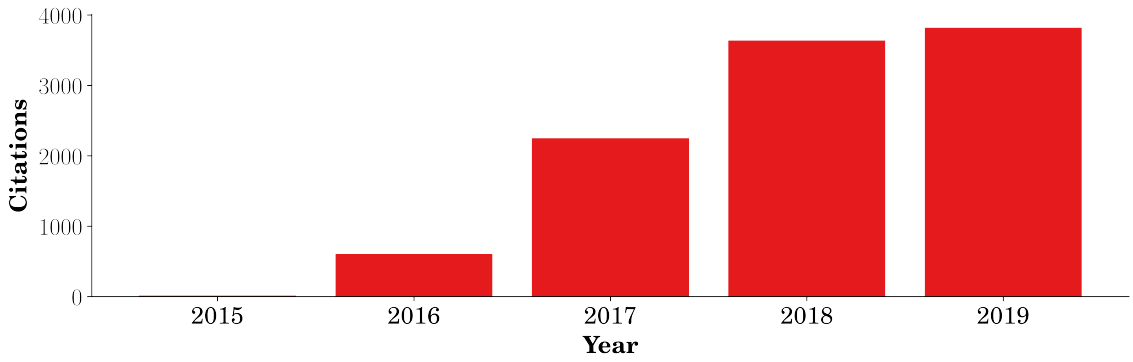


Figure 4.2: Number of citations for the TensorFlow Python library [77]

4.1.1 Machine Learning for Health Monitoring

With advanced software packages such as TensorFlow and Scikit-Learn, machine learning inference is increasingly easy and effective to deploy on low-power devices. With such edge computing capability so easily realized in a health monitoring application, significant benefits can be realized [78]. These benefits include reduced bandwidth requirements for data transmission as the edge devices transmit model inference results rather than raw data. In addition, edge inference reduces latency for health monitoring applications due to reduced reliance on network resources.

Recently, a variety of machine learning methods have been used to characterize bearing health [79]. To support this effort, it is typically necessary to perform 'feature extraction' — that is, determine the quantitative *features* from a data set which are believed to be informative towards developing the desired predictive model. For example, time-domain statistical features may be extracted to train Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Self-Organizing Maps (SOM) to characterize various bearing faults [80]. Empirical mode decomposition has been used to extract features for ANNs to predict bearing failure modes [81].

A more abstract machine learning paradigm, called deep learning, relies on complex model architectures to automatically perform feature extraction steps, thereby limiting

the need for manual data modification [82]. Deep learning has revolutionized the fields of machine vision, gaming, and language processing through high-level abstraction and shows promise as a tool for health monitoring applications in the future [83]. In an IoT architecture, deep learning in the cloud may induce excessive latency due to bandwidth and storage constraints. For this reason, edge computing has been suggested as a means of streamlining data flow and protecting user privacy [84]. However, deep learning models are 'black boxes' in the sense that their predictions are often difficult or impossible to delineate from a human perspective. This lack of transparency can be problematic when attempting to make predictions in fields where human safety is at stake [85].

4.2 Classification

Classification is an example of supervised learning, wherein a model is trained to find some most likely class $\hat{y}$, associated with a set of evidence, $E = (x_1, x_2, \ldots, x_n)$. This may be formalized by

$$\hat{y} = \underset{y}{\operatorname{argmax}} f(E), \tag{4.1}$$

where f is a function of the evidence, E. Typically, the elements in E are referred to as the *features* of the dataset. This is simply an abstract term for the mathematical values which constitute the data. The features may be any combination of the parameters which describe the data, and must typically be down-selected by an engineer based on the desired model outcomes.

When considering classification algorithms for health monitoring, it is important to recognize that these statistical models require training data which include labeled healthy and unhealthy data. This is a significant hurdle to overcome, as manufacturing equipment is designed to operate as reliably as possible. Creating an experimental setup and performing run-to-failure experiments in a controlled environment is not a trivial task and may take weeks to perform [63]. In a production environment, it may take years to capture and label

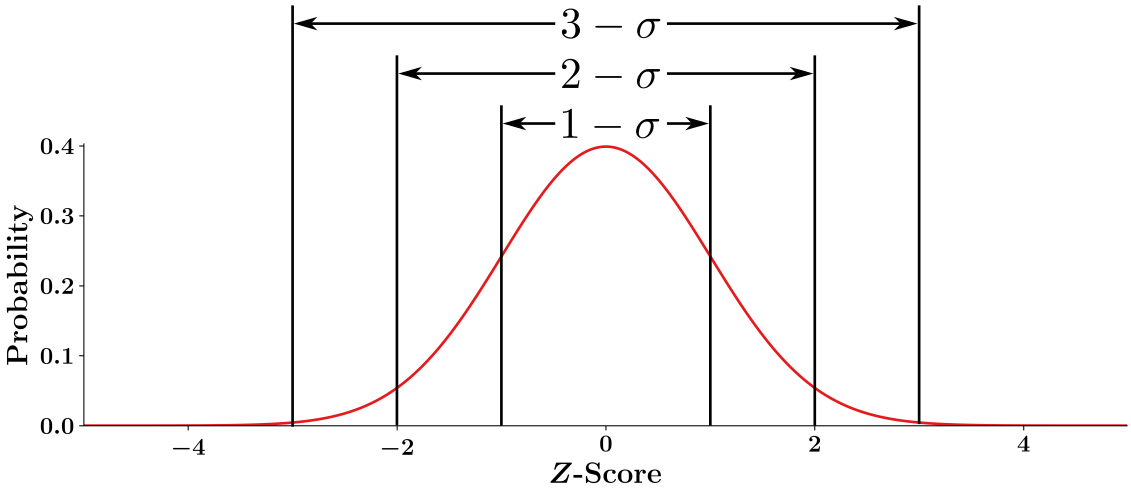


Figure 4.3: Normal distribution

a sufficiently varied set of healthy and fault data from machinery.

4.3 Novelty Detection

Novelty detection, a type of unsupervised learning, is a fundamentally different approach to health monitoring. The objective of a novelty detection algorithm is to correctly determine when a data stream reveals underlying changes in a process which were not present at the time of model training. Given the incredibly open-ended nature of the topic, a wide variety of methods have been studied in this line of research [86].

4.3.1 Control Charts

Statistical control charts are a well-studied and widely used tool for tracking the health of a machine or manufacturing process over time, and an application of novelty detection. Indeed, maintaining statistical control of a process is a principal objective of Lean Six Sigma — a ubiquitous framework for continuous process improvement in the manufacturing industry [87]. The concept of statistical process control is based around the Normal distribution, shown in Figure 4.3. This is a special case of the Gaussian

Table 4.1: Sigma level versus capture rate

Sigma Level	Capture Rate	Outliers per Million
1σ	69%	308538
2σ	99.38%	6210
3σ	99.9996%	3.4

Distribution, defined by

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \tag{4.2}$$

where σ is the standard deviation and μ is the mean of the distribution. In the Normal distribution, the standard deviation is set to $\sigma = 1$ and the mean is $\mu = 0$. Data which follow this distribution fall within some multiple of σ at rates defined by Table 4.1. For example, the $\pm 1\sigma$ range will capture 69% of all data, corresponding to over 3×10^3 outliers per million. If a manufacturing process is normally distributed and a specification mandates that the process falls within this range, an unacceptably high number of parts will be considered out of specification. In Lean Six Sigma, the objective is to improve the process such that the specification limits correspond to $\pm 3\sigma$ of the process variation.

Ideally, a random sample of process data exhibits variation which can be normalized with a static mean and standard deviation. By normalizing a sample of process data in this way, a wide array of statistical tools can be readily employed to assess whether these data are in a state of control or not. One such tool is the set of simple guidelines called the Nelson Rules [88]. Essentially, these rules are simple, statistically derived ways of determining whether new data can be confidently associated with a defined probability distribution.

4.4 Statistical Methods

While a huge variety of model types are available for classification and novelty detection methods, statistical methods are straightforward and readily explainable. Two such models

— the Gaussian Naive Bayes classifier and the Gaussian Mixture Model — are briefly discussed in this section.

4.4.1 Gaussian Naive Bayes

The Gaussian Naive Bayes classifier (GNB) is a simple, statistically-derived classification method. As a specific type of classifier, the objective of GNB is to find a predicted class, $\hat{y}$ such that

$$\hat{y} = \underset{y}{\text{argmax}} P(y) \prod_{i=1}^{n} P(x_i|y) \tag{4.3}$$

where

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{4.4}$$

is Gaussian and μ_y, σ_y^2 are the learned means and variances of the class, respectively. This model is "Naive" in the sense that it assumes all features, x_i, are independent. While this assumption almost never holds in practice, the Naive Bayes model compares well in classification accuracy against more sophisticated methods [89]. For this reason, this classifier is used as a baseline in this work.

4.4.2 Gaussian Mixture Models

Similar to the GNB classifier, Gaussian Mixture Models (GMM) attempt to learn a Gaussian distribution which best fits a provided set of data. As it does not require labeled data, this is an unsupervised learning method and can be used to facilitate novelty detection.

4.5 Artificial Neural Networks

Artificial Neural Networks (ANN) are biologically inspired mathematical models known for their ability to model any function. While ANNs have existed in concept since the 1960's, they only grew to prominence in the last decade or so. At their most basic level, ANNs are a highly simplified analog of biological neurons. Their basic mathematical

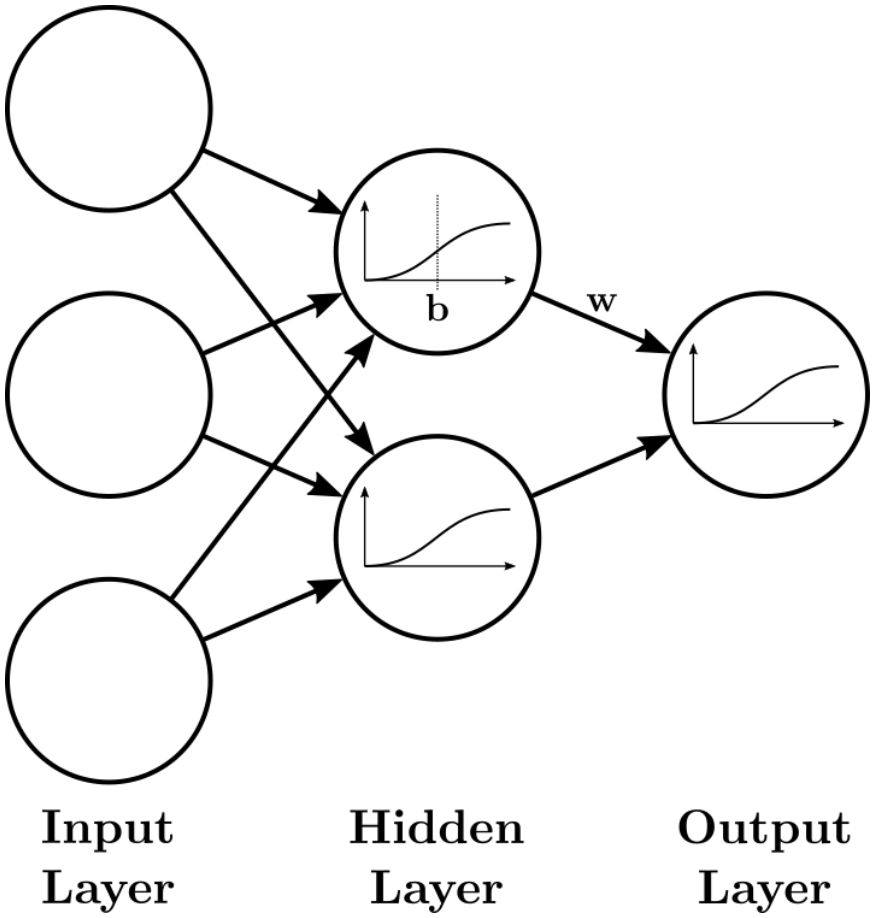


Figure 4.4: Artificial Neural Network structure

components are shown in Figure 4.4. The individual circles in this image represent "neurons," connected to one another so that the input from a neuron in one layer influences those in the next layer and so on. This figure shows a neural network with three layers — Input, Hidden, and Output. Three neurons in the input layer represent three input features. The connection between each neuron is weighted by a parameter, w. Furthermore, each neuron past the input layer has an activation function, $\sigma(x)$, shifted by some bias, b.

In order to yield a desired output provided some input data, neural networks are trained by modifying their weights and biases via the backpropagation algorithm [90]. This is an optimization problem wherein a cost function, typically the error between the desired and actual outputs of the neural network, is minimized. This is done by determining the gradients of each parameter with respect to the cost function and iteratively updating them

until the algorithm terminates.

4.6 Dimensionality Reduction and Latent Representation

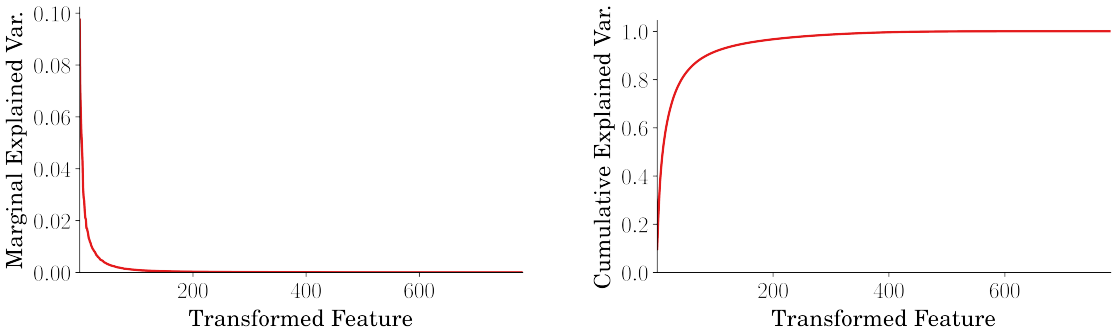
4.6.1 Principal Components Analysis

Principal Components Analysis (PCA) is a linear dimensionality reduction technique used to retain maximum variance while projecting a dataset to a lower dimensional subspace. Generally speaking, PCA generates some transformed matrix T which arranges the n features of decreasing variance for an $m \times n$ dataset X which contains m samples. If the columns of X are normalized to have zero mean, this transformation can be readily expressed as

$$T = XW, \tag{4.5}$$

where W is an $n \times n$ matrix which has columns defined as the eigenvectors of $X^T X$. With the original dataset transformed by (4.5), it is possible to choose an arbitrary number of components, p to retain while keeping a desired percentage of the variance from the full dataset. In doing so, the dimensionality of the original data is reduced based on the difference between the original and reduced number of columns $n - p$. PCA is an excellent preprocessing step in machine learning. Its basis in linear algebra makes it deterministic and easy to visualize.

To demonstrate the use of PCA for dimensionality reduction, consider the MNIST handwritten digits dataset [91]. This is a widely used set of 70,000 handwritten digits. Each digit is a 28×28 pixel image and therefore contains 784 dimensions. The task of analyzing this dataset and creating accurate classifiers for it is a common "hello world" in machine learning. PCA can be used to easily compress the dimensionality from these data with minimal loss. Figure 4.5 shows the marginal and cumulative explained variance of the PCA decomposition for the entire dataset. This is a measure of the fraction of variance in the data captured by each principal component. Clearly, the first hundred or so dimensions



(a) Marginal explained variance

(b) Cumulative explained variance

Figure 4.5: PCA of the MNIST dataset

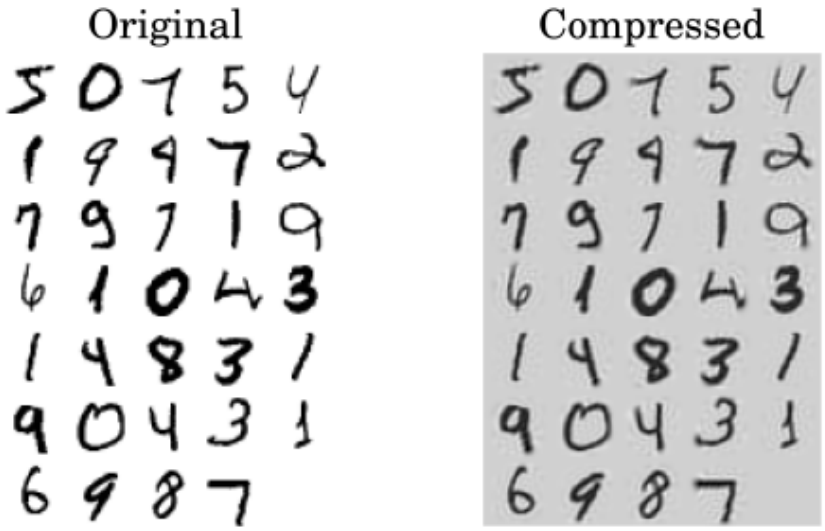


Figure 4.6: Loss due to PCA dimensionality reduction - first 154 components

are much more important than the rest.

The effectiveness of PCA on this dataset is shown in Figure 4.6. In this image, a random sample of the original digits is shown on the left. From this sample, PCA was used to extract the first 154 components, which captures 95% of the variance. The 154 dimensional digits are reconstructed and shown on the right side of the figure. Although the compressed digits look almost identical to the originals, they occupy 20% of the memory.

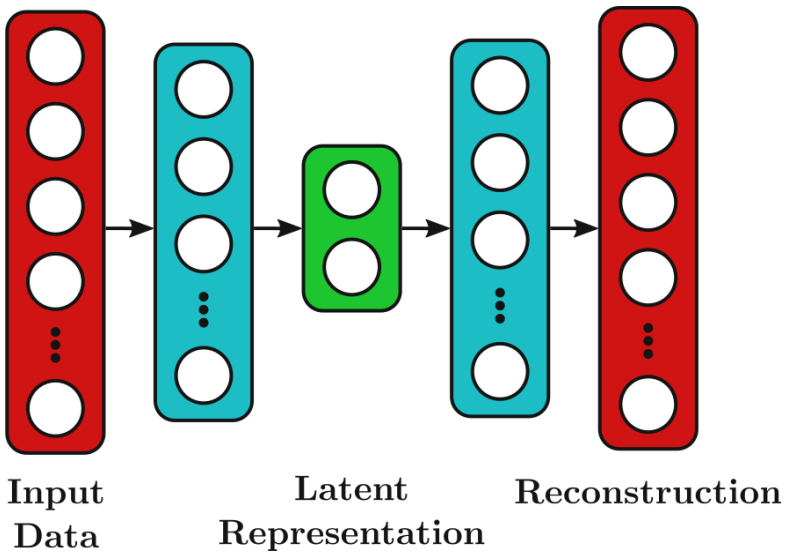


Figure 4.7: Typical autoencoder architecture

Table 4.2: Autoencoder for MNIST data

Layer	**Output Shape**	**No. Parameters**
Input	28×28	0
Flatten	784	0
Dense	100	78,500
Dense	30	3,030
Dense	100	3,100
Dense	784	79,184
Reshape	28×28	0
Total Parameters:	163,814	

4.6.2 Autoencoders

While PCA is an excellent tool for machine learning and statistical analysis, it is inherently limited due to its linearity. Autoencoders are neural networks which can be applied more abstractly by using nonlinear activation functions and/or multiple layers. A representation of an autoencoder architecture is shown in Figure 4.7. An autoencoder accepts some input and attempts to find a mapping in a low-dimensional space from which that input can be reconstructed with minimal error.

To illustrate the use of a basic autoencoder, consider the network architecture shown

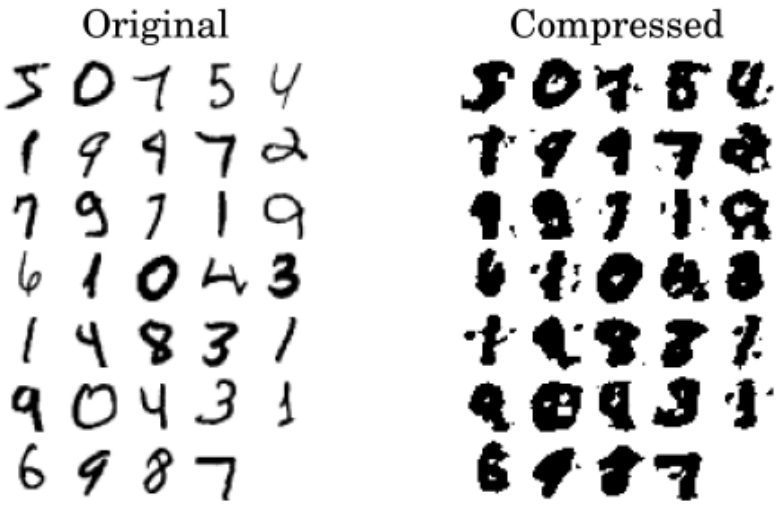


Figure 4.8: Loss due to autoencoder dimensionality reduction - 30-dimensional subspace

in Table 4.2. This is a relatively simple structure with one hidden layer and a latent space with 30 degrees of freedom. After training this neural network on the same MNIST data, the original and compressed representation of these digits are shown in Figure 4.8. While these digits generally have similar shape to the originals, substantial loss has occurred in the compression process. Again, neural networks are highly sensitive to hyperparameter selection such as number of hidden layers, optimizer parameters, and number of nodes per hidden layer.

For an image compression task, the use of convolutional layers makes more intuitive sense than the multi-layer perceptron architecture. An example architecture is summarized in Table 4.3. This convolutional autoencoder compresses the original images into several 3×3 images before reconstructing them from this latent space. By using convolutional filters instead of artificial neurons, this architecture requires significantly fewer parameters. In addition, it works substantially better than the multi-layer perceptron autoencoder, as summarized in Figure 4.9. The compressed digits very closely resemble the originals, demonstrating the efficiency of convolutional layers in generalizing features for this task.

Table 4.3: Convolutional Autoencoder for MNIST data

Layer	**Output Shape**	**No. Parameters**
Input	28×28	0
Convolution (2D)	$28 \times 28 \times 16$	160
Batch Normalization	-	-
Dropout	-	-
Max Pooling (2D)	$14 \times 14 \times 16$	-
Convolution (2D)	$14 \times 14 \times 32$	4,640
Batch Normalization	-	-
Dropout	-	-
Max Pooling (2D)	$7 \times 7 \times 32$	-
Convolution (2D)	$7 \times 7 \times 64$	18,496
Batch Normalization	-	-
Dropout	-	-
Max Pooling (2D)	$3 \times 3 \times 64$	-
Convolution Transpose (2D)	$14 \times 14 \times 16$	4,624
Convolution Transpose (2D)	$28 \times 28 \times 1$	145
Total Parameters:	46,753	

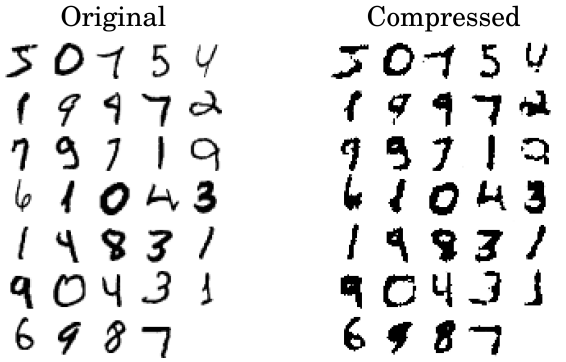


Figure 4.9: Loss due to Convolutional autoencoder dimensionality reduction

4.7 A Case Study with Simulated Data

Using modern statistical and machine learning toolkits, each of the previously mentioned tools can be readily deployed in an IoT framework, whether on an edge device or in the cloud. To demonstrate the application of machine learning and statistical tools in an

Table 4.4: Healthy and Unhealthy Train Dataset Parameters

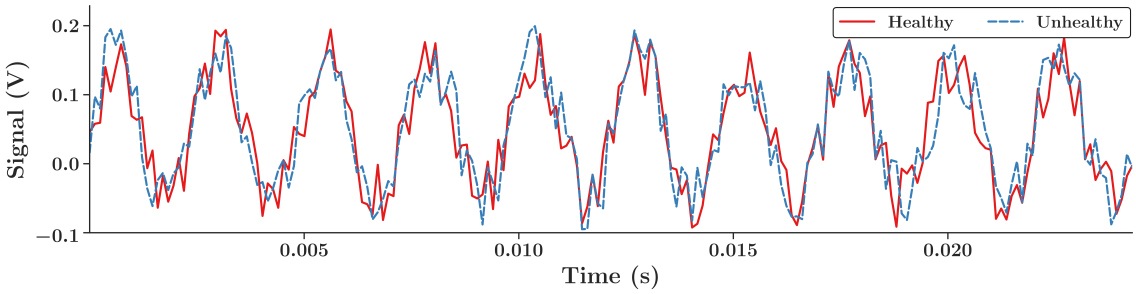
Parameter	**Value**
Amplitude	$[0.1, 0.37, 0.63, 0.9, 1.167, 1.43, 1.7, 1.97, 2.23, 2.5]$
Frequency	$[409.6, 500.6, 591.6, 682.7, 773.7, 864.7, 955.7, 1046.8, 1137.8, 1228.8]$

Table 4.5: Healthy and Unhealthy Validation Dataset Parameters

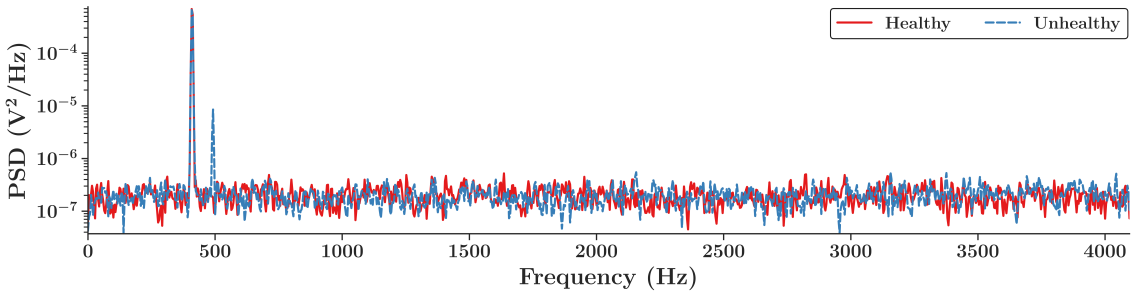
Parameter	**Value**
Amplitude	$[0.23, 0.48, 0.73, 0.98, 1.24, 1.49, 1.74, 1.20, 2.25, 2.5]$
Frequency	$[452.7, 538.9, 625.2, 711.4, 797.6, 883.9, 970.1, 1056.3, 1142.6, 1228.8]$

IoT framework, a training dataset is generated which represents a significantly simplified vibration signal from a piece of factory equipment. These signals contain 1 second of data sampled at 8,192 Hz. Gaussian white noise of variance $\sigma = 0.1$ is added to each time series. The amplitudes and frequencies of these signals are provided in Table 4.4. Each frequency and amplitude combination are replicated 7 times. This set of signals represents vibration from a healthy machine. For an 'unhealthy' example, the same dataset is repeated with additional frequencies and amplitudes defined by multiplying the baseline data by 1.4 and 0.1 respectively. This additional frequency represents a defect frequency present in the signal. Similarly, a validation dataset using the frequency and amplitude values shown in Table 4.5 is generated to evaluate the generalizability of the various machine learning algorithms. These validation values cover a similar range of amplitudes and frequencies and a good statistical model should exhibit similar performance for both the validation and training datasets.

The test data are shown in Figure 4.10. The time-series of both healthy and unhealthy signals are shown in Figure 4.10a. From the time series, these signals seem effectively identical due to the inclusion of noise and the low amplitude of the defect frequency. Only by looking at the Power Spectral Density of these signals in Figure 4.10b can the differences between these signals be easily spotted. While the amplitude of the defect frequency is quite low, it is still significantly above the noise floor.



(a) Time-series signal



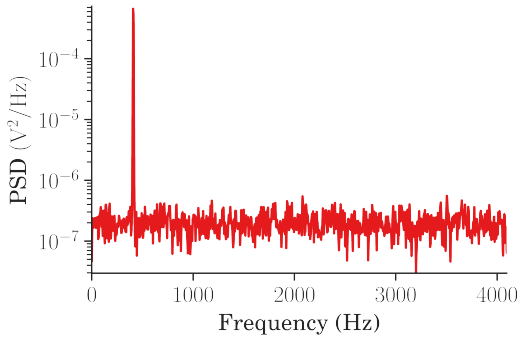
(b) Power Spectral Density

Figure 4.10: Example of test dataset

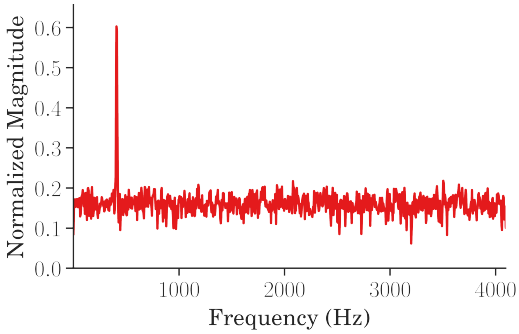
Using this test dataset, it is possible to demonstrate the use of machine learning algorithms to detect the presence of the defect frequency and classify the health status of a signal.

4.7.1 Signal Preprocessing

For any statistical or machine learning application, data must be preprocessed before they may be used for model building and inference. This is especially true for data such as the PSD. The mean value of the Power Spectral Density plot is on the order of 1×10^{-7}. In general, this mean value may vary wildly depending on background noise, other frequencies present, and other things. For these data to be useful, they must be normalized. Many machine learning methods and neural network activation functions are designed to accommodate data values in the range $[0, 1]$. While many methods can be used to normalize a dataset to this range, a log-linear interpolation is chosen for these frequency-domain features. This method ensures that the normalized features remain



(a) Original Power Spectral Density



(b) Normalized Power Spectral Density

Figure 4.11: Log-Linear normalization for frequency content

interpretable to an engineer and meet the formatting requirements for common machine learning algorithms. This normalization is demonstrated in Figure 4.11. The original PSD plot is shown in Figure 4.11a. To ensure high resolution for the frequency content, these data are plotted on a logarithmic scale on the y-axis. The normalized version of these data are shown in Figure 4.11b. The mapping used for this normalization is a log-linear scale from $(1 \times 10^{-8}, 1 \times 10^{0})$ to $(0, 1)$.

4.7.2 Novelty Detection

Using the preprocessed test data, machine learning methods can be compared for their ability to detect outliers which may be indicative of health degradation. Several methods of varying complexity are used to compare their strengths and weaknesses as well as the computational power required to implement them. First, PCA is combined with Gaussian Mixture Modeling to perform dimensionality reduction and learn the nominal distribution of the reduced-order features. This "PCA-GMM" model is expected to be efficient and easily computed due to its ability to be implemented in the Scikit-Learn Python library. However, its use of PCA for feature extraction will likely render this model unable to generalize well to the validation data. For the second and third anomaly detection models, TensorFlow is used to create autoencoders for anomaly detection. First, a multi-layer perceptron autoencoder (AE) demonstrates the effectiveness of very simple neural network

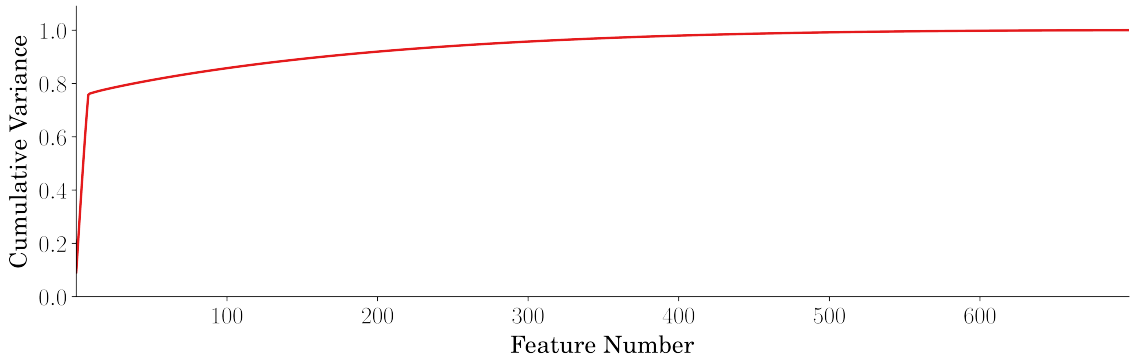


Figure 4.12: Cumulative explained variance from PCA composition

architectures for this type of anomaly detection problem. This is expected to be an inferior neural network architecture for anomaly detection. Next, a Convolutional Autoencoder (CNN-AE) is used to demonstrate the effectiveness of Convolutional layers for this type of problem. Due to the efficiency of convolutional networks in describing features with a small number of parameters, this architecture is expected to be computationally efficient while generalizing well to the validation data.

PCA is used to determine the dimensionality of the latent subspace for the PCA-GMM model. Given the task is anomaly detection, the normalized frequency-domain features from the healthy training samples are used. In all, 700 randomized samples, each with 1,024 features constitute this training set. After performing PCA on the training data, the cumulative explained variance is shown in Figure 4.12. In this plot, two trends are apparent: one section for the first eight features where additional features greatly increase the cumulative explained variance and another where additional features have much lower influence on the explained variance. To avoid building models which fit to noise, the first 64 principal components will be used. This 64-dimensional subspace accounts for approximately 80% of the variance for the test dataset. With this parameter established, it is straightforward to fit a Gaussian Mixture Model to the compressed training data of shape 700×64.

Similarly, a 64-dimensional latent subspace is chosen for the AE model. This choice

Table 4.6: Autoencoder for Example Data – AE model

Layer	**Output Shape**	**No. Parameters**
Input	1,024 × 1	0
Dense	512	524,800
Dense	64	32,832
Dense	512	33,280
Output	1,024 × 1	525,312
Total Parameters:	1,116,224	

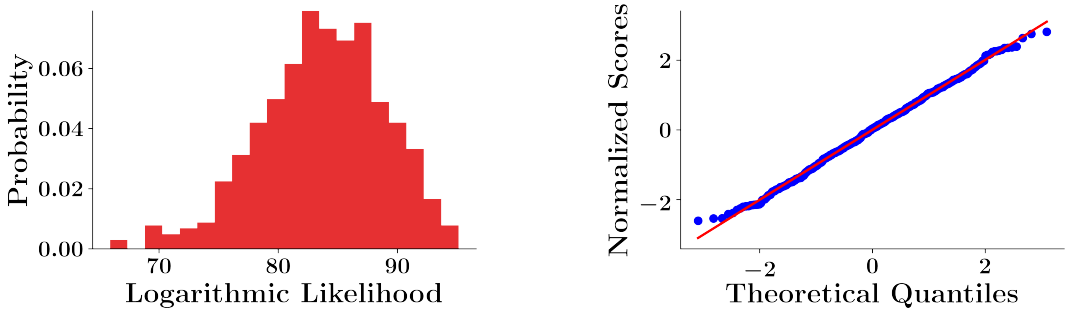
of latent dimensionality ensures a fair comparison with the PCA-GMM model; both are operating with the same amount of compressed data. The architecture for this autoencoder model is summarized in Table 4.6. As shown, this model has one intermediate layer between the latent space to extract the relevant features. This is quite a large model relative to the PCA-GMM; the AE contains 1.1 million parameters which, when trained on a GPU-accelerated PC, can finish training within one minute.

Finally, the CNN-AE model does not utilize the same latent space that the previous models do, so it is not easy to directly compare the dimensionality of the latent space for a fully convolutional autoencoder to the other models. The architecture for the CNN-AE model is shown in Table 4.7. While the multiple convolutional, pooling, normalization, and dropout layers appear excessively complex, the efficiency of convolutional neural networks is apparent in the small number of parameters — 3,289 — necessary to train this autoencoder. This is orders of magnitude smaller than the AE model and slightly smaller than even the PCA-GMM model.

To assess the performance of each model in both detecting unhealthy signals and generalizing from the training data to the validation data, all three models are benchmarked and trained on the same datasets. Specifically, each model is trained to learn a representation of the healthy training data. Based on this learned representation, these models are compared for their ability to statistically distinguish the healthy and unhealthy data while not falsely detecting a difference between the training and validation data.

Table 4.7: Convolutional Autoencoder Example Data

Layer	**Output Shape**	**No. Parameters**
Input	1,024 × 1	0
Convolution (1D)	1,024 × 8	32
Batch Normalization	-	-
Dropout (50%)	-	-
Max Pooling (1D)	512 × 8	-
Convolution (1D)	512 × 16	400
Batch Normalization	-	-
Dropout (50%)	-	-
Max Pooling (1D)	256 × 16	-
Convolution (1D)	256 × 24	1,176
Batch Normalization	-	-
Dropout (50%)	-	-
Max Pooling (1D)	128 × 24	-
Up Sampling (1D)	256 × 24	-
Convolution (1D)	256 × 16	1,168
Up Sampling (1D)	512 × 16	-
Convolution (1D)	512 × 8	392
Up Sampling (1D)	1,024 × 8	-
Convolution (1D)	1,024 × 1	25
Total Parameters:	3,289	

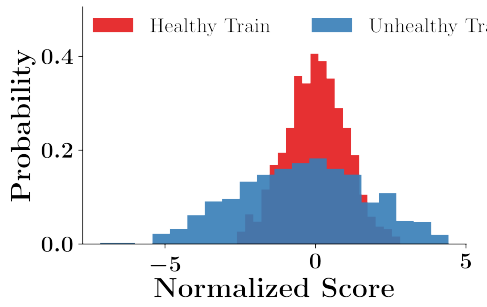


(a) Histogram of training data

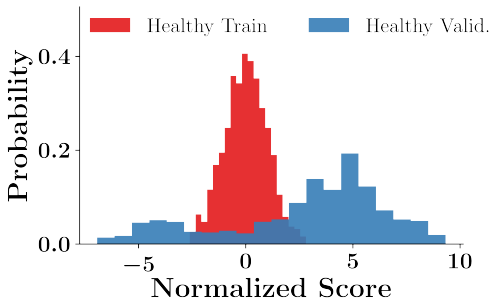
(b) Normalized scores

Figure 4.13: PCA-GMM likelihoods for healthy training data

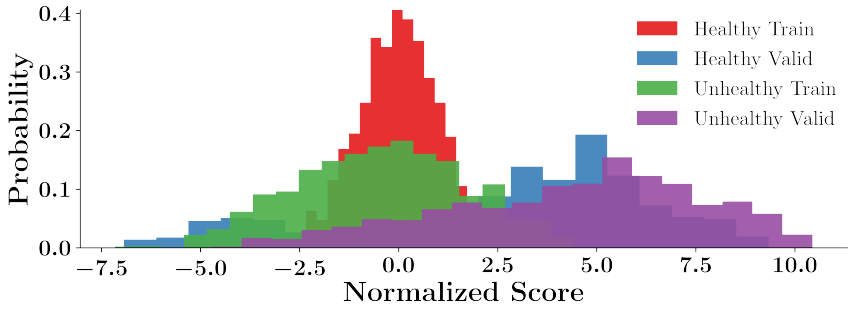
Because the PCA-GMM model is probabilistic, logarithmic likelihood values are used to assess the probability that a specific sample is associated with the model. These values, based on the healthy training data, are summarized in Figure 4.13. The histogram of the logarithmic likelihoods is shown in Figure 4.13a. As is clear in this figure, the distribution



(a) Healthy vs Unhealthy training data



(b) Healthy Training vs Validation data

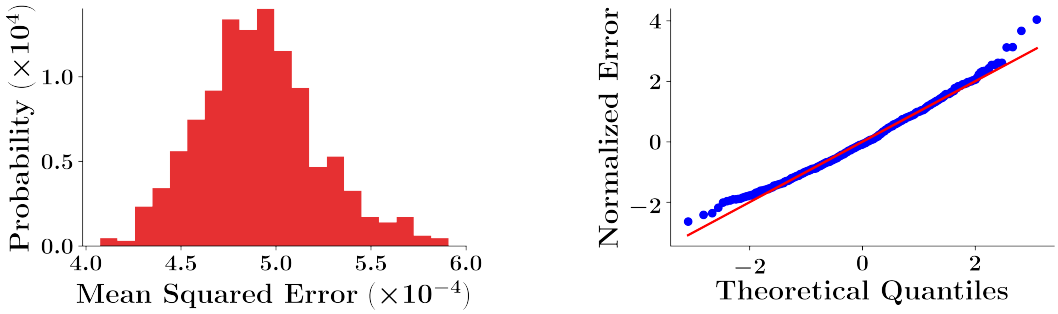


(c) Comparison of all data

Figure 4.14: PCA-GMM normalized scores

is skewed with a long tail towards low likelihoods. Based on the assumption that this distribution is log-normal, the normalized distribution is shown in Figure 4.13b. Using a χ^2 test, this normalized distribution has a p-value of 0.40, indicating that it is, indeed, normal.

With a normalized training dataset, it is possible to evaluate the performance of the PCA-GMM model relative to the healthy, unhealthy, and validation data. This comparison is summarized in Figure 4.14. First, it is important to be able to detect differences in the healthy and unhealthy training data. The histogram in Figure 4.14a shows that while the unhealthy scores have a negligible mean shift, the variance is substantially higher relative to the healthy scores. Similarly, Figure 4.14b indicates a substantial mean shift and increase in variance from the healthy training scores to the validation scores. The full comparison of all data is shown in Figure 4.14c. Here, it is clear that, while a difference in distribution



(a) Histogram of training data

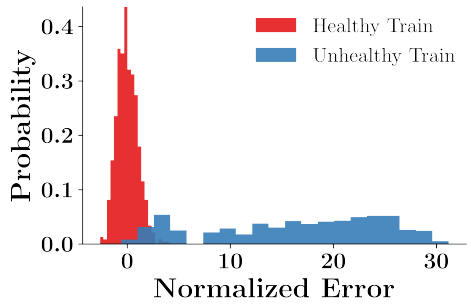
(b) Normalized Scores

Figure 4.15: AE mean squared error for healthy training data

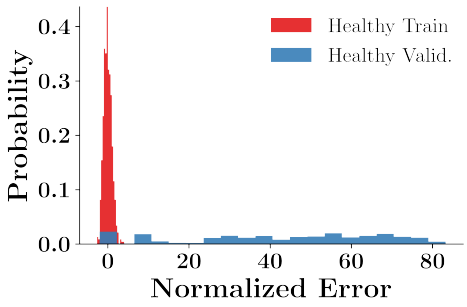
is apparent between the healthy and unhealthy training scores, the PCA-GMM model does not generalize well to the validation data.

In a similar fashion, the AE model is trained to correctly reconstruct the spectrogram of the healthy training data. After training, the performance of the AE model is summarized in Figure 4.15. The probability distribution of the mean squared error for reconstructing the training data is shown in Figure 4.15a. Although this distribution does not meet the statistical significance test for being Gaussian ($p \approx 0.001$), it is treated as Gaussian for the sake of this discussion. With this assumption, the distribution can be easily normalized, as shown in Figure 4.15b.

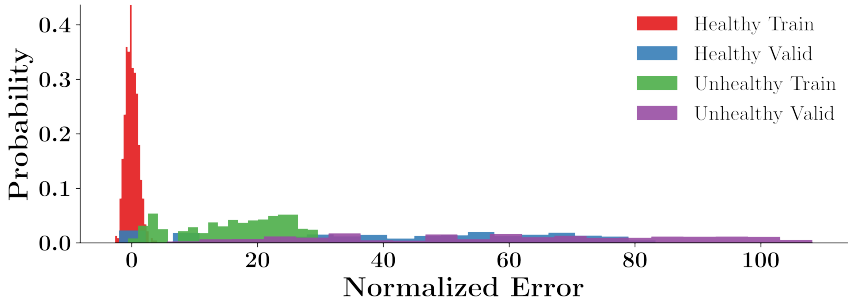
With the mean squared error in reconstructing the healthy training data normalized, the AE model may be analyzed for how it reconstructs the unhealthy and validation data. These results are presented in Figure 4.16. Similarly to the PCA-GMM model, the AE model can successfully distinguish between the healthy and unhealthy training data, as shown in Figure 4.16a. The vast majority of the unhealthy data fall well outside of the 3σ range, and can easily be flagged as anomalies relative to the normalized scores from the healthy data. However, this model falls victim to the same issue as the PCA-GMM model in failing to generalize beyond the healthy data. As Figure 4.16b shows, the healthy validation data are even more widely distributed than the unhealthy training data. The full comparison of the normalized errors is shown in Figure 4.16c.



(a) Healthy vs Unhealthy training data

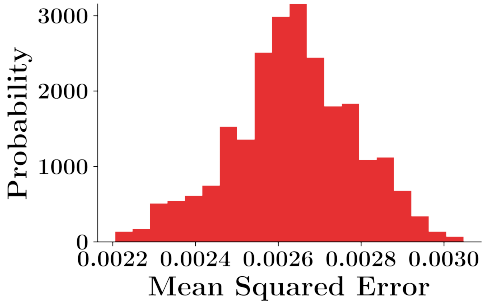


(b) Healthy Training vs Validation data

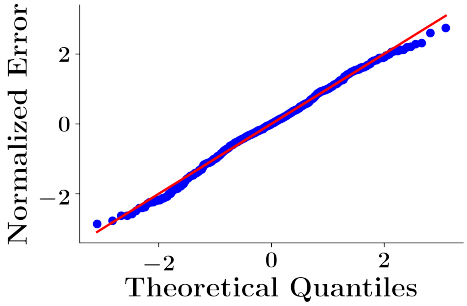


(c) Comparison of all data

Figure 4.16: AE normalized errors



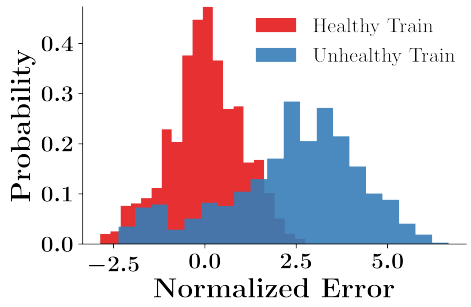
(a) Histogram of training data



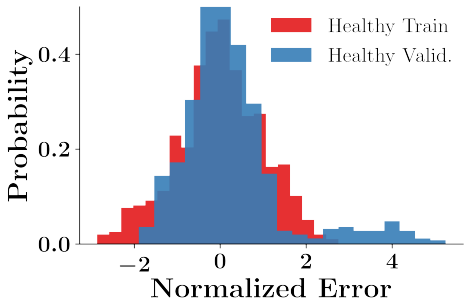
(b) Normalized Scores

Figure 4.17: CNN-AE mean squared error for healthy training data

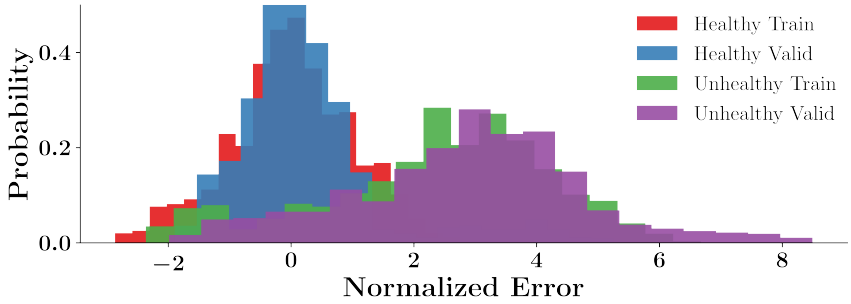
The final model used in this anomaly detection analysis is the CNN-AE. Unlike the AE model, this neural network uses convolutional layers to extract features from the training data. After training the CNN-AE model, the reconstruction errors for the healthy data are shown in Figure 4.17. Again, the distribution of mean squared error is shown



(a) Healthy vs Unhealthy training data



(b) Healthy Training vs Validation data



(c) Comparison of all data

Figure 4.18: CNN-AE normalized errors

in Figure 4.17a. In this case, a statistical test weakly indicates that this distribution is Gaussian ($p \approx 0.05$). With a Gaussian assumption, the normalized distribution is shown in Figure 4.17b.

As with the previous models, the CNN-AE model is compared for performance with validation and unhealthy data in Figure 4.18. Similarly to previous models, this model shows excellent separation between the healthy and unhealthy training data in Figure 4.18a. The unhealthy training data have a normalized mean of $\mu_{unhealthy} \approx 8.0$ and will be readily distinguishable from the healthy data. Unlike previous models, the CNN-AE also shows good generalization to the validation data as shown in Figure 4.18b. By using convolutional layers for feature extraction, this model readily recognizes the pattern of a single frequency in a given range, regardless of its specific location. The full comparison is shown in Figure 4.18c. As is expected, the healthy and unhealthy training and validation data are

approximately superimposed with one another.

In the analysis of three different models for anomaly detection in the frequency-domain, it is clear that the CNN-AE model is superior to the PCA-GMM and AE models. In this application, the convolutional layers are exceptionally useful in identifying features regardless of their specific location, while PCA and vanilla neural networks tend to learn very specific representations based on their training data.

4.7.3 Classification

As another approach to health monitoring, statistical classification algorithms are commonly used to identify healthy and unhealthy data. Using classification methods presented in this chapter, a series of models are designed to classify healthy and unhealthy signals from statistical features in the training and validation datasets. Again, three such classification models are used to compare their performance and ease of deployment in an IoT and edge inference framework. First, PCA is combined with a Gaussian Naive Bayes to form a statistical model, PCA-GNB, which is deployed in the lightweight Scikit-Learn framework. Due to the large feature space of the initial data, this model is expected to generalize rather poorly. Next, a Multi-Layer Perceptron (MLP) neural network is implemented in TensorFlow. Finally, a similar neural network with convolutional feature extraction layers — CNN-MLP — is chosen to compare the use of convolutional layers in a classification framework.

In a manner similar to the PCA-GMM model, Principal Components Analysis is used to generate features from the normalized frequency spectrum in the PCA-GNB model. These features are then used to create a Gaussian Naive Bayes classifier to distinguish healthy from unhealthy data. Again, the first 64 principal components are used for the classification model. The Gaussian Naive Bayes classifier is trained on the entire training set of 1,400 samples and evaluated against both the training and validation data. Because it is not generally possible to expose a GNB classifier to validation data without explicitly

Table 4.8: Multilayer Perceptron Classifier for Example Data – MLP model

Layer	**Output Shape**	**No. Parameters**
Input	1,024 $\times$ 1	0
Dense	512	524,800
Dense	128	65,664
Dense	64	8,256
Output	2	130
Total Parameters:	598,850	

including that data in the learning model, it is expected that the PCA-GNB model will perform poorly on the validation data.

The multilayer perceptron (MLP) model is summarized in Table 4.8. Unlike the MLP-AE model, this neural network is designed with two outputs to accommodate a confidence metric for healthy and unhealthy data. As this model does not attempt to reconstruct the input signal, it is approximately half as large as the MLP-AE model with just under 600,000 parameters. Again, this baseline model is used to demonstrate the capabilities of a neural network in classifying healthy and unhealthy frequency data.

Similarly, the CNN-MLP model is summarized in Table 4.9. This model contains three convolutional layers with dropout, normalization, and max pooling followed by three dense layers. These final MLP layers are identical to those in the MLP model. By using convolutional layers for preprocessing, it is expected that this model will generalize better on the validation data than the MLP model alone. In addition, these layers do not add many parameters to the overall size of the model.

To assess the performance of the classifier models, they are each fit to the training data set and tested on the validation data. The summary of each classifier model performance is shown in Table 4.10. The True Positive Rate, defined by

$$\mathrm{TPR} = \frac{tp}{tp + fn} \tag{4.6}$$

where tp and fn represent the true positive and false negative predictions respectively,

Table 4.9: Convolutional Classifier Example Data

Layer	**Output Shape**	**No. Parameters**
Input	1,024 $\times$ 1	0
Convolution (1D)	1,024 $\times$ 8	8
Batch Normalization	-	-
Dropout (50%)	-	-
Max Pooling (1D)	512 $\times$ 8	-
Convolution (1D)	512 $\times$ 8	200
Batch Normalization	-	-
Dropout (50%)	-	-
Max Pooling (1D)	256 $\times$ 8	-
Convolution (1D)	256 $\times$ 8	8
Batch Normalization	-	-
Dropout	-	-
Max Pooling (1D)	128 $\times$ 8	-
Flatten	1,024	-
Dense	512	524,800
Dense	64	32,832
Dense	2	130
Total Parameters:	558,290	

indicates the model's capacity to correctly capture all of the positive results. Similarly, the True Negative Rate, defined by

$$\mathrm{TNR} = \frac{tn}{tn + fp} \tag{4.7}$$

where tn and fp represent the true negative and false positive predictions, respectively, indicates the model's capacity to correctly capture all negative results. Ideally, a model will approach 100% TPR and TNR. The neural network models both have the capacity to measure their performance against the validation data during training without explicitly utilizing these data for computations. As a result, they have a slight advantage over the PCA-GNB model. With this in mind, it is noteworthy that the MLP model performs worse than the PCA-GNB model across all validation metrics. By comparison, the CNN-MLP model exhibits relatively high validation accuracy of 83.9%. From these results, it is clear

Table 4.10: Classifier Model Performance Summary

Model Name	Train Accuracy	Validation Accuracy	True Positive Rate	True Negative Rate
PCA-GNB*	94.1%	63.6%	79.2%	59.2%
MLP	99.4%	57.5%	70.6%	54.2%
CNN-MLP	99.6%	83.9%	100%	75.8%

* - Not exposed to validation data during training

Table 4.11: Model Deployment Options

Name	RAM	Operating System	Location	Cost
Desktop	64GB	Ubuntu 18.04	Local	$1,200
Amazon-EC2	8GB	Ubuntu 18.04	Cloud	$35/month
PocketBeagle	512MB	Debian 9	Edge	$27
BeagleBone Black	512MB	Debian 9	Edge	$78

that a more complex model such as a MLP classifier does not inherently provide advantages over a simple GNB classifier. It is also clear that the addition of convolutional layers is helpful in classification.

4.7.4 IoT and Edge Deployment

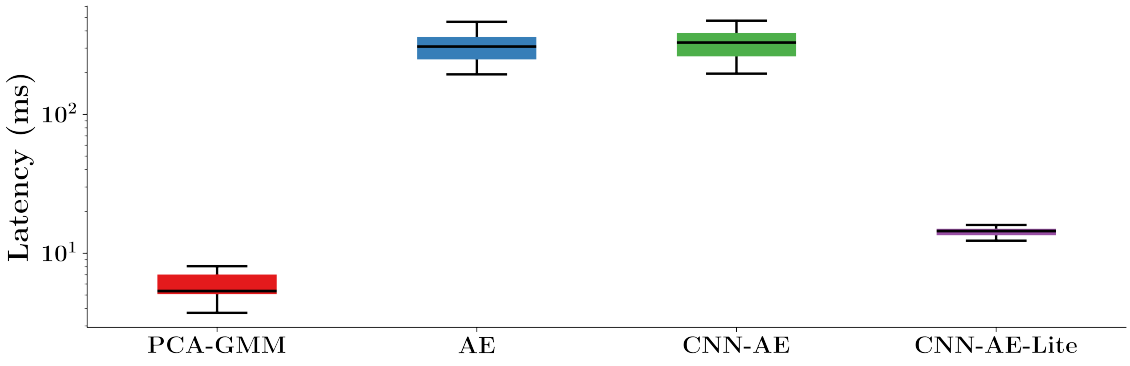
When developing machine learning algorithms for health monitoring, it is crucial to consider the practical deployment of these tools. While many past research efforts have developed novel and useful health monitoring algorithms, their deployment is often overlooked. For the simulated data and corresponding algorithms presented in previous sections, a number of deployment options are compared for their efficiency. These options are summarized in Table 4.11. A desktop computer with an AMD Ryzen 7 3700x CPU, 64GB of DDR4 RAM, and an NVIDIA Geforce 2060 GPU serves as a baseline for personal computing performance. This is compared to an Amazon EC2 instance with 2 'Virtual CPUs' and 8GB of RAM running in the cloud. In an attempt to control for variations due to operating systems, both the Cloud and Local computers utilize Ubuntu 18.04, a

Table 4.12: Deployed Models

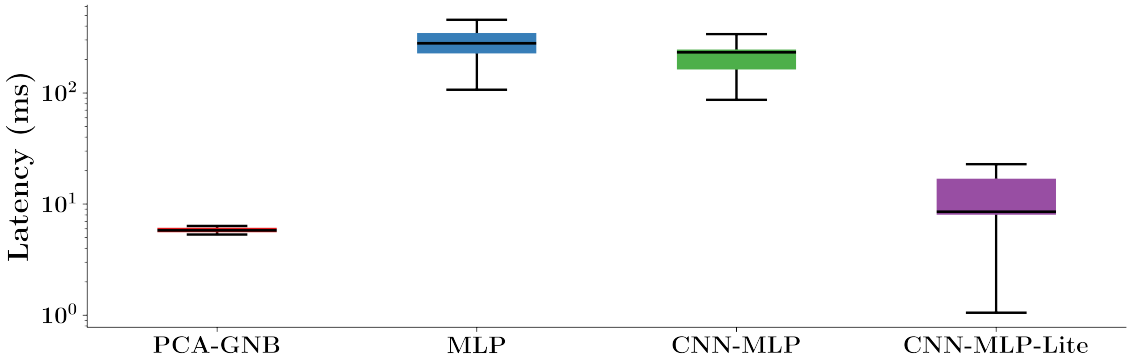
Name	Buffer Size	Basis Library	Notes
PCA-GMM	367kB	Scikit-Learn v0.22.2	-
AE	13.4MB	TensorFlow v2.0.0	Edge Deploy Excluded
CNN-AE	125kB	TensorFlow v2.0.0	Edge Deploy Excluded
CNN-AE-Lite	139kB	TensorFlow v2.0.0	-
PCA-GNB	274kB	Scikit-Learn v0.22.2	-
MLP	6.43MB	TensorFlow v2.0.0	Edge Deploy Excluded
CNN-MLP	50.8MB	TensorFlow v2.0.0	Edge Deploy Excluded
CNN-MLP-Lite	16.9MB	TensorFlow v2.0.0	-

Linux-based OS. These machines have state-of-the-art commercially available computing capabilities for personal and cloud use. In addition, two very similar embedded computers — the PocketBeagle and BeagleBone Black are evaluated in this study. These low-power devices lack the advanced CPU architectures and RAM of the other machines, but their low cost and GPIO pins allow them to be used as integrated data acquisition and model inference devices.

Due to memory and computational limitations, the AE, CNN-AE, MLP, and CNN-MLP models cannot directly be run on the BeagleBone and PocketBeagle. However, TensorFlow possesses the capability of compressing most model architectures into a 'Lite' format. This compression can be leveraged to deploy neural networks onto microcontrollers with kilobytes of RAM [74]. While deploying machine learning to embedded devices is a growing field in and of itself, very simple steps can be taken to prepare neural networks for deployment on Linux devices such as the BeagleBone and PocketBeagle. In this study, the CNN-AE and CNN-MLP models are compressed into this format to demonstrate the ease in computational load at the cost of insignificant performance loss. To evaluate the performance of each deployment option, all of the anomaly detection and classification models are saved as serial buffers in their respective formats. The size of these serial buffers is proportional to the number of parameters in the respective model. In all, eight models are deployed to the various computing devices. The properties of each model are



(a) Anomaly detection model latency



(b) Classification model latency

Figure 4.19: Desktop deployment computation latency

summarized in Table 4.12. The full TensorFlow models, while excluded on the BeagleBone and PocketBeagle, are deployed on the Desktop and Amazon EC2 instance to compare those platforms. The Scikit-Learn and TensorFlow Lite models are deployed to the edge devices to compare the computational load of these models on limited hardware.

First, the computational requirements for each model are compared on the Desktop. Results from this analysis are summarized in Figure 4.19. The computational latency for the anomaly detection models are shown in Figure 4.19a. Note here that the y-axis is shown on a logarithmic scale in milliseconds. By using this scale, variations in runtime of the more efficient models are kept visible while allowing a comparison with the full neural network models which require significantly more time to run. Clearly, the PCA-GMM model exhibits the lowest computational latency at less than 10ms on average. By comparison,

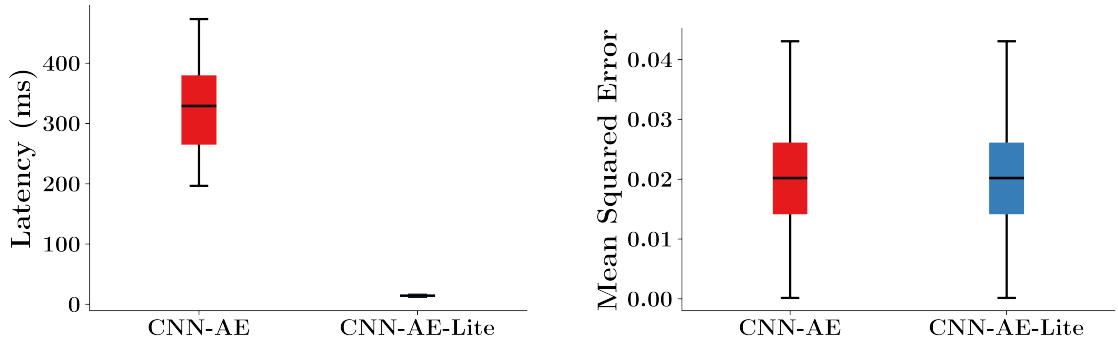
Table 4.13: Lite versus Full Model Performance Summary

Name	Mean Latency Decrease (ms)	Max Latency Decrease (ms)	Mean Absolute Output Error	Mean Squared Error
CNN-AE-Lite	313	884	1.87×10^{-10}	2.23×10^{-8}
CNN-MLP-Lite	201	507	1.70×10^{-10}	2.38×10^{-8}

both of the TensorFlow models require several hundred milliseconds to run. Finally, the TensorFlow Lite model offers a substantial reduction in computation time relative to the full CNN-AE, while still requiring roughly twice as much time as the PCA-GMM. Very similar results for the classification models are summarized in Figure 4.19b. In a departure from the trends in the previous figure, the CNN-MLP-Lite model exhibits significant variation in runtime relative to the CNN-AE-Lite model. This variation — on the order of 10ms — may be due in part to the non-deterministic nature of the Linux operating system. In such non real-time systems, it is difficult to control for timing of computing tasks on the millisecond level and lower.

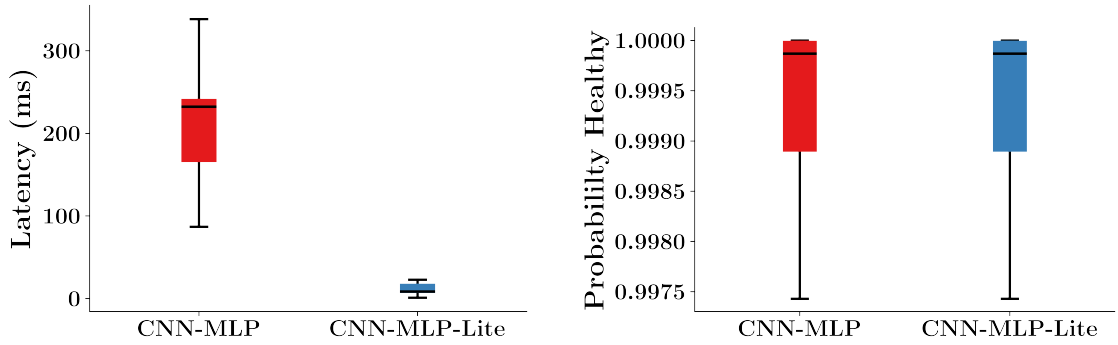
While the Lite models require substantially less computation time than the full TensorFlow models, they also do not perform any worse than the full models in their tasks of classification or signal reconstruction. Figure 4.20 highlights the performance of the CNN-AE Lite and full models relative to one another. The latency comparison is repeated in Figure 4.20a to illustrate how much more computationally efficient the Lite model is. Not only is it an order of magnitude quicker to compute, but the variance is also much smaller relative to the full model. This benefit is very useful in a real-time application, where the inference bandwidth is determined by the worst-case latency. The comparison of reconstruction error for both models is shown in Figure 4.20b. For this specific example, The full and Lite models perform identically. These results compare similarly to those of the CNN-MLP model, summarized in Figure 4.21.

A detailed comparison of the Lite and full TensorFlow models is provided in Table 4.13. This table summarizes the latency comparisons and output error of the Lite models relative



(a) Computation time (b) Model output

Figure 4.20: CNN-AE lite versus full model comparison

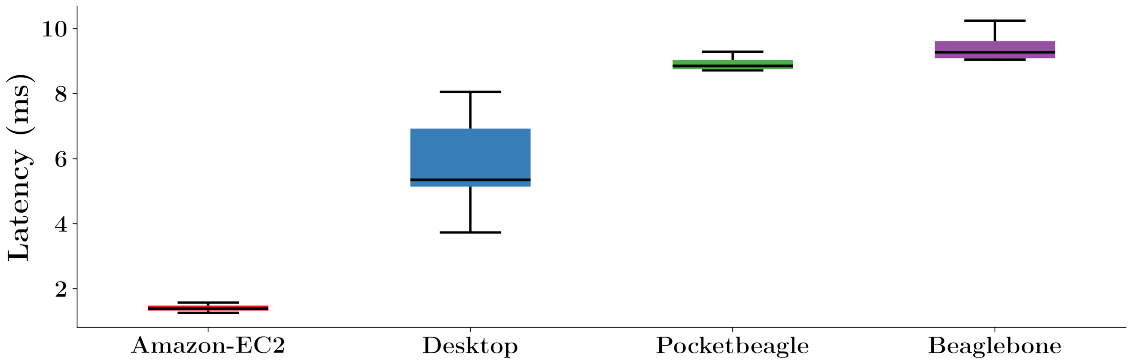


(a) Computation time (b) Model output

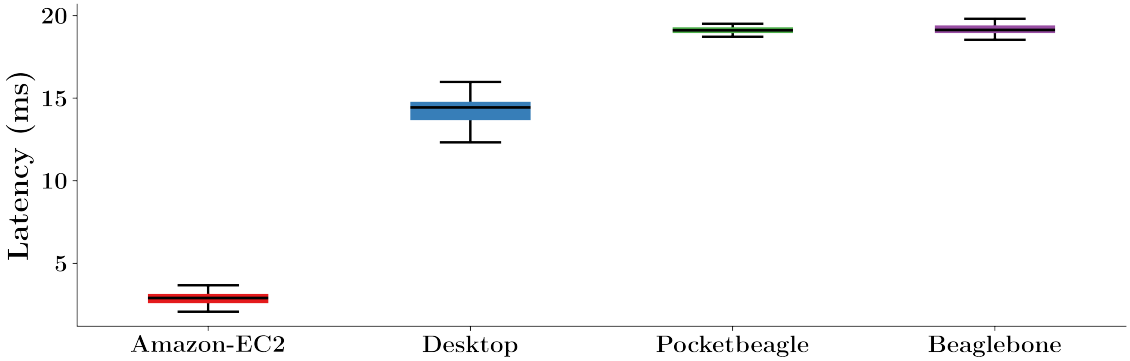
Figure 4.21: CNN-MLP lite versus full model comparison

to the full model. For both neural network models, the Lite version requires several hundred milliseconds less computation time on average, with maximum time savings approaching a full second. At the same time, the mean absolute error for the healthy training data is on the order of 1×10^{-10}, with the mean squared error for the whole dataset at 2×10^{-8}. This is a remarkable result; with minimal effort and no loss of precision, these neural network models can be compressed to a format which runs substantially more quickly.

While the previous discussion confirms that machine learning algorithms can efficiently run on the Desktop deployment, the efficiency of different deployment options for specific models is shown in Figure 4.22. The computation latency comparison for the PCA-GMM model is shown in Figure 4.22a. This Scikit-Learn model runs with exceptional efficiency on the Desktop as previously discussed, taking roughly 6ms on average to run. When



(a) PCA-GMM Latency



(b) CNN-AE-Lite latency

Figure 4.22: Computation latency for various deployment options

deployed to both the BeagleBone Black and PocketBeagle, the model takes slightly longer at approximately 9ms. Considering the cost difference and edge deployability of these embedded computers, this is a trivial increase in latency compared to the Desktop. Finally, the Amazon-EC2 instance is the most efficient deployment option at under 2ms of computational latency. This is likely a testament to the optimization of the cloud computing platform; Amazon boasts "burst" computing power which may explain the very low computational latency of this deployment option. A similar trend is apparent in the CNN-AE-Lite latency comparison shown in Figure 4.22b. This model requires approximately twice as long to run on all platforms when compared to the PCA-GMM model. However, a computation time of 20ms is still remarkably fast on a low-power edge device such as the PocketBeagle. For both models, it is clear that they could be deployed in a near real-time application based on these data. It is worth noting that these latency values

only report the required time for each device to perform inference for each model. While the Amazon-EC2 deployment is the clear winner in this category, it operates in the cloud. Therefore, this increased performance due to computational latency is quickly lost through network latency. This relationship is discussed more thoroughly in the next chapter

4.8 Conclusion

This chapter has presented several tools for rapid deployment of modern machine learning models in a machine health monitoring application. With recent advances in open-source, permissively-licensed software, a great variety of models may be deployed to the edge or cloud at little cost. When compared to deployment on modern personal computing hardware, embedded and cloud-based systems can perform inference with a minimal increase in latency.

This chapter also introduced a methodology for preprocessing and building models to assess the health of frequency-domain data. By normalizing the PSD of a signal end ensuring consistent features, high-quality modelling tools can readily be applied to assess the health of a signal. The next chapter will address the methods available to deploy these models in an edge device for near real-time data acquisition and model inference.

CHAPTER 5

AN OPEN-SOURCE, INTEGRATED DATA ACQUISITION EDGE DEVICE

This chapter describes the design of an integrated data acquisition device which utilizes open-source, low-cost hardware and software components. Relying on the open-source BeagleBone platform, this data acquisition device leverages real-time capable hardware and advanced machine learning software to incorporate sensor data acquisition and analysis in a single device. When compared to powerful cloud-based computing for vibration data analysis, the proposed data acquisition platform performs more reliably as the data are kept locally.

5.1 Advancement of Embedded and Open-Source Tools

Recent advancements and contributions from the open-source community have facilitated the use of powerful IoT tools in manufacturing environments. For example, a multitude of IoT platforms are currently available for implementation, many with open-source code and permissive licensing [92]. One such platform, used in this study, is Node Red. This lightweight, user-friendly tool can be easily deployed on a server or IoT device to facilitate communications between devices. This platform includes built-in functionality to use common web tools such as MQTT and REST services. Beyond this, Node Red, can be used to populate and query databases, interact with embedded devices, and extract CNC data.

5.1.1 Low-Cost Electronics

For many years, the lack of readily-available distributed computing power, signal processing software, and accurate, low-cost sensors rendered low-cost data acquisition tools technically and economically infeasible. Within the last decade, many changes on

all of these fronts have substantially changed the landscape of distributed data acquisition. First, recent years have seen an explosion in low-cost, widely accessible embedded hardware. Open Source embedded Linux boards such as the BeagleBone and Raspberry Pi, combined with true embedded system architectures such as the Arduino are at the forefront of this advancement. Each of these hardware platforms feature unprecedented computing power with advanced ARM processors. Additionally, many of these platforms include a pinout suite with tools such as Analog-to-Digital Converters (ADC), UART, I2C, and SPI. With these tools, a user can easily integrate with an increasingly wide array of analog and digital sensors designed to operate at low voltage.

These low-power sensors have inherent limitations compared to their industry standard counterparts. First, low-voltage analog sensors have limited measurement range and resolution; a 30-Volt analog sensor can output a signal with 6 times the range of a 5-Volt sensor. Second, these sensors do not typically feature embedded signal processing functionality, such as analog filters, afforded to their high-cost counterparts. Low-cost sensors also do not feature standardized, "plug-and-play" interfaces. The user must instead design and fabricate a board with the necessary signal and power routing to connect to the main processor. In spite of these shortcomings, low-cost sensors can be used in low-to-medium range signals with acceptable performance. For instance, accelerometer development boards with bandwidths below 10kHz are readily available for less than $\$100$. Such an accelerometer can be connected to an ADC pin on an Arduino-compatible microcontroller for rapid prototyping.

Finally, the availability of software architectures which expedite the development of embedded platforms has seen significant advancement in recent years. The true strength of systems such as the Raspberry Pi, BeagleBone, and Arduino lie in the ever-growing communities developing and sharing software under permissive, "Copyleft" licenses such as the GNU General Public License.

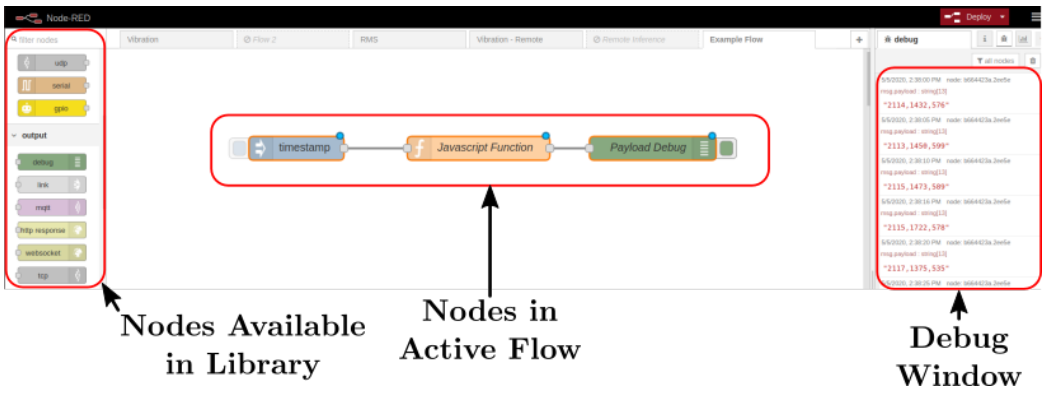


Figure 5.1: Node Red dashboard

5.1.2 Node Red

Node Red is a JavaScript-based tool for rapidly prototyping and deploying IoT technologies. Originally developed by IBM in 2013, Node Red is an open source, royalty-free software package which can be easily integrated into any IoT deployment [93]. Like the Python machine learning libraries mentioned in the previous chapter, Node Red is distributed under the permissive Apache 2.0 license, which allows for patentable derivative works. As an in-browser flow-based programming platform, Node Red can be installed in the cloud, on embedded Linux devices, or on a personal workstation.

An example of the developer user interface for Node Red is shown in Figure 5.1. In the middle of the screen, a window shows the current flow. Here, nodes are placed and connected to one another to relay messages in a desired manner. In this flow, a simple "inject" node is connected with a JavaScript function which simply has an output to the debug window. With an open-source framework, a multitude of node options are available to be downloaded and used in the Library pictured on the left side of the screen. Serial communication, MQTT inputs and outputs, HTTP requests, and database integration are some examples of the types of nodes available through Node Red. The inclusion of generic "JavaScript function" nodes and "Executable" nodes gives the user freedom to execute arbitrary JavaScript code from the browser as well as call external executable files on the Node Red server as well. As a result, Node Red is an excellent tool for rapidly prototyping

and deploying code in the Internet of Things.

5.2 BeagleBone

The Beagleboard.org Foundation is a community-driven non-profit credited with the development of the BeagleBone embedded hardware architecture [94]. This hardware platform is used in a number of open-source, single-board Linux computers as shown in Figure 5.2. The most common, general-purpose BeagleBone is the BeagleBone Black. This single-board computer operates with a version of Debian Linux and is capable of performing a wide variety of tasks with a large array of GPIO pins. Other versions of this board include the low-power PocketBeagle and the BeagleBone AI. The specifications of each version are summarized in Table 5.1. From this table, it is clear that the PocketBeagle and BeagleBone Black are approximately identical from a computational power standpoint. By comparison, the BeagleBone AI is a more powerful, less mature platform intended to

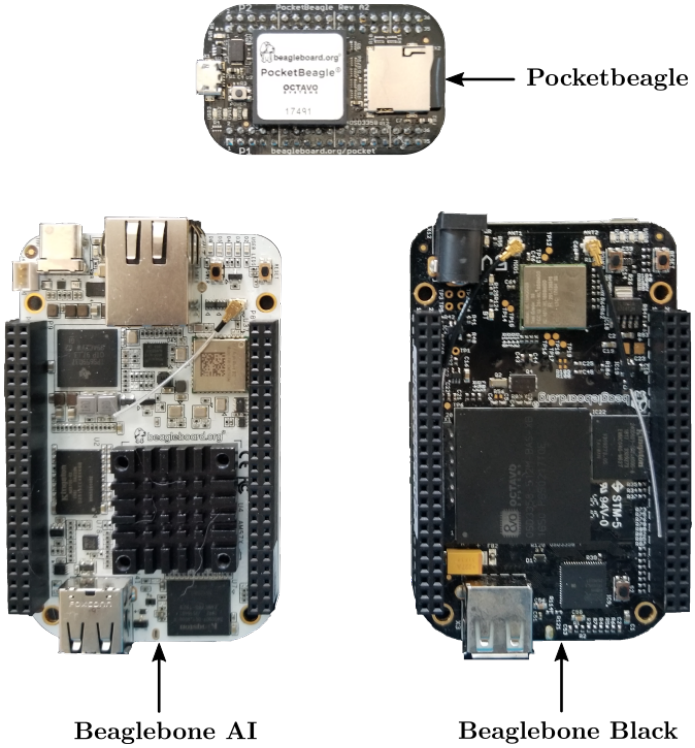


Figure 5.2: BeagleBone computers

Table 5.1: BeagleBone Specification Comparison

Board	Processor	RAM	Flash Storage
BeagleBone Black	1GHz ARM® Cortex-A8	512MB DDR3	4GB
PocketBeagle	1GHz ARM® Cortex-A8	512MB DDR3	None
BeagleBone AI	2x dual ARM® Cortex®-M	1GB DDR3	16GB

bring machine learning to the BeagleBone platform.

While the BeagleBone Black possesses impressive computational power for a small footprint, other embedded Linux computers such as the Raspberry Pi outperform it in general-purpose computing. The real strength of the BeagleBone resides in its integration with two ARM A335x microprocessors called Programmable Realtime Units (PRUs). These computing chips are designed to execute basic commands at high clock rates with sub-microsecond precision. Coupled with a 12-bit Analog-to-Digital Converter (ADC) with 100kSPS bandwidth, the BeagleBone is readily capable of performing high-frequency analog measurements. In addition, this device possesses a large suite of General-Purpose Input/Output (GPIO) pins which the PRUs can access for read and write instructions.

As an open-source computer platform, the BeagleBone is frequently used by hobbyists for robotic applications. However, the powerful input-output capabilities of the BeagleBone make it suitable for professional applications and deployment in consumer designs as well. For example, the PocketNC is a desktop 5-axis CNC machine which is controlled by the BeagleBone Black and its PRUs for real-time, precise control of several stepper motors [95]. Similarly, a startup company Creator uses the BeagleBone for control of an automated hamburger cooking robot [96]. The BeagleBone Black is also at the center of a Polymerase Chain Reaction machine produced by Chai [97]. These companies represent a growing list of innovators which rely on the flexibility and control provided by modern embedded Linux computers.

5.3 Data Acquisition

To facilitate machine health and process monitoring, this platform is designed to accommodate both CNC controller and sensor data. Two CNC machine protocols — MTConnect and OPC-UA — are considered in this work. The inclusion of these protocols demonstrates the flexibility of Node Red. These communication standards also cover a large percentage of modern CNC machines. When considering sensor data acquisition, a focus is given to vibration sampling. In reality, a large suite of thermocouples, current sensors, motion sensors, etc. can be incorporated into this platform. Within this range of sensor types, accelerometers are unique in the density and quantity of data that they ordinarily produce. As such, any data acquisition strategy which can successfully accommodate accelerometer data can be easily transferred to other analog sensors which create much less data.

5.3.1 MTConnect

With the use of Node Red, the proposed data acquisition platform is capable of capturing controller data from machines using the MTConnect protocol. A representative schematic of this data acquisition process is shown in Figure 5.3. At the bottom of the image, the machine controller and its MTConnect adapter are shown. The data acquisition device leverages Node Red to continuously sample the MTConnect adapter via HTTP. The machine returns an XML payload with all of the current data. These data are then parsed according to the specifications of the digital architecture and placed in JSON payloads. Finally, the JSON payloads are published to the MQTT message broker with their associated topic.

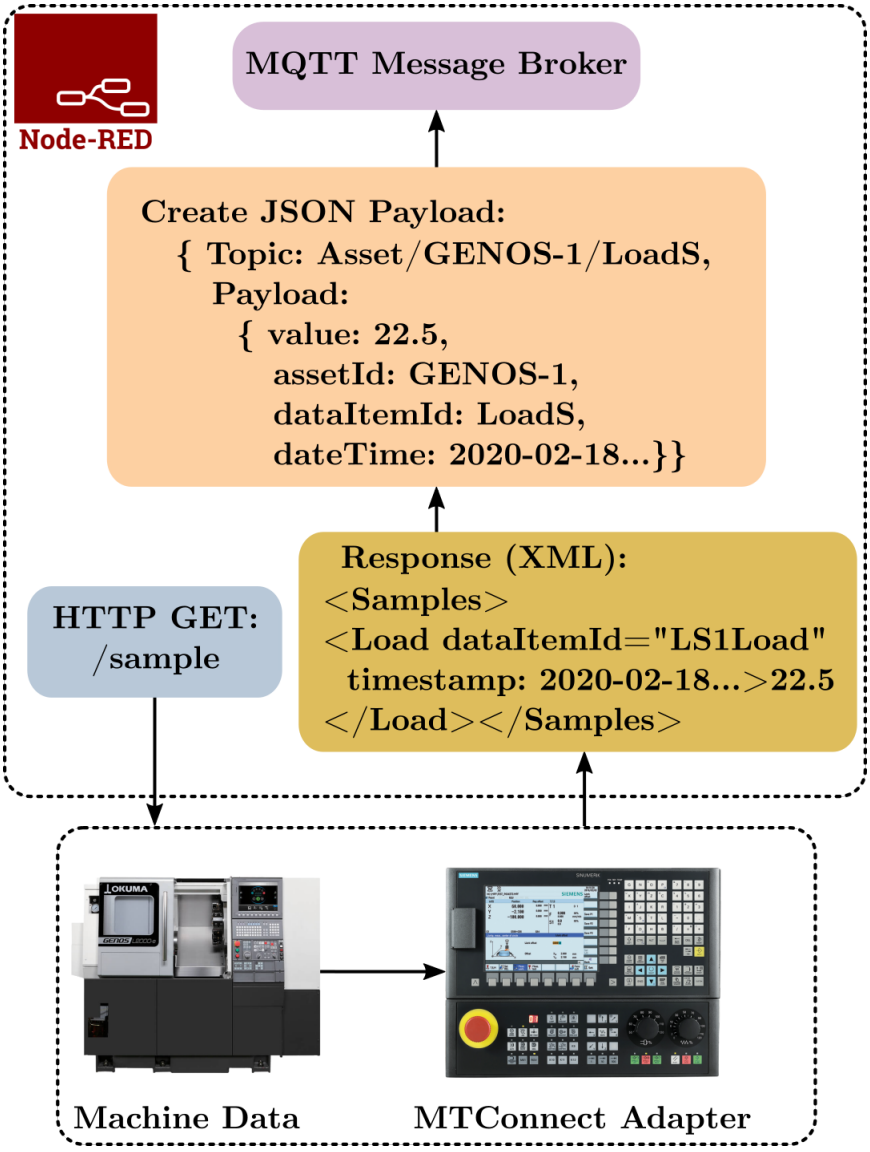


Figure 5.3: Schematic of MTConnect data acquisition

5.3.2 OPC-UA

In a similar fashion, the data acquisition device is capable of capturing OPC-UA data from a CNC machine. The process of retrieving OPC-UA data is slightly different from that of MTConnect data, as shown in Figure 5.4. With this protocol, an inject node requests specific data items from the OPC-UA server at a given interval. For example, this figure illustrates reading the *aaLoad[4]* address every 0.5 seconds. The OPC-UA server then sends a JSON response with the data item name as the topic and the current measurement

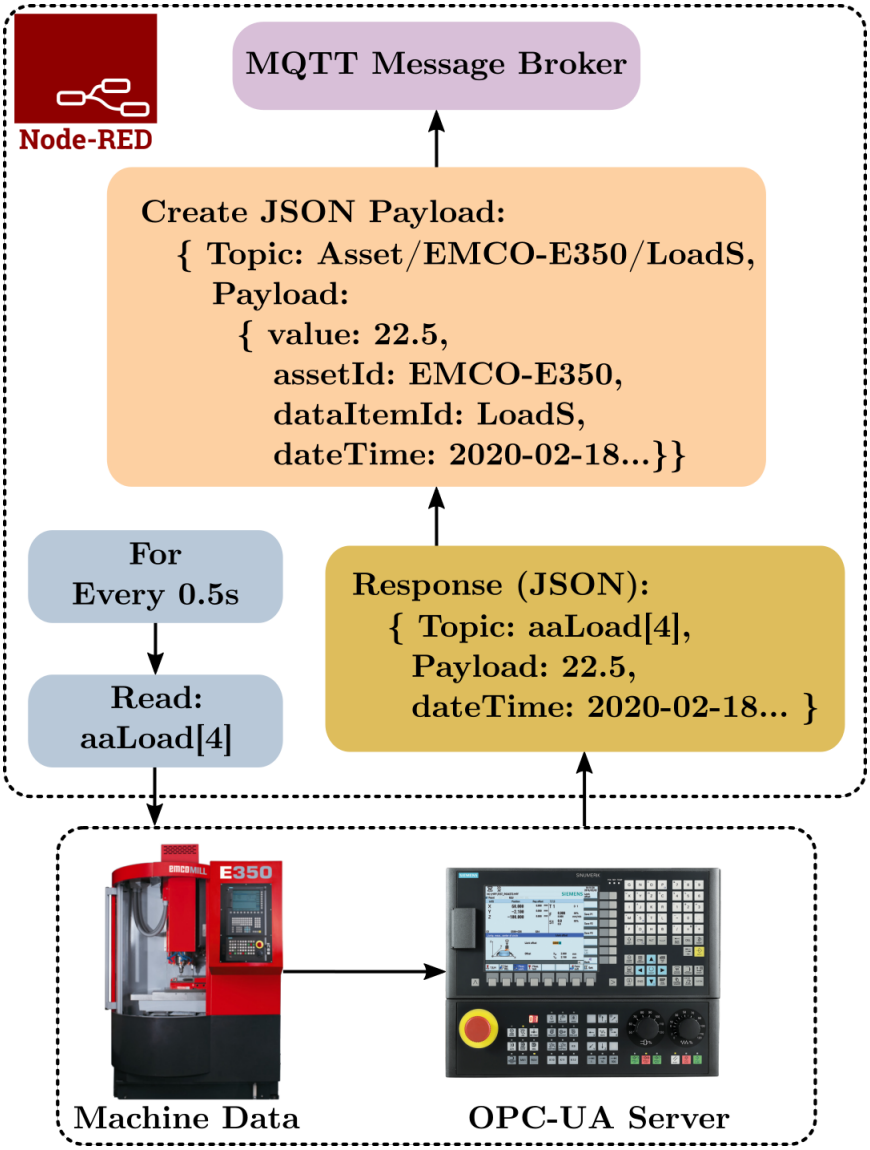


Figure 5.4: Schematic of OPC-UA data acquisition

as the payload. This response is then translated into a format which complies with the digital architecture prior to being sent to the MQTT message broker.

OPC-UA servers are designed with maximum sampling rates for each data tag individually and a total overall bandwidth. For example, the Siemens 828D can accommodate up to 100 simultaneously monitored data items. While each data item may have a maximum sample rate of 20Hz, it is not possible to sample all 100 data items at this rate. To reliably accommodate several simultaneous data items, a sample rate of 2Hz

is chosen in this work. This low rate is insufficient to capture some potentially useful data items such as motor loads, especially if no filtering is done prior to sampling from the OPC-UA server. However, for basic process parameter data acquisition such as the current program name and spindle speed, this is a sufficient refresh rate.

5.3.3 Analog Sensors

While any generic computer could perform the CNC controller data acquisition presented above, the BeagleBone platform is unique in its ability to perform onboard sampling of analog sensors. Using the PRUs, this device is capable of recording high-precision analog vibration measurements and passing these data along in an IoT architecture. A basic schematic of this process is shown in Figure 5.5. With multiple analog input pins available, the BeagleBone can capture data from multiple sensors if desired. These measurements are read by the PRU and stored in a ring buffer before being written to a raw serial file in the Linux userspace. The data from this file can then be imported into Node Red and published to the MQTT message broker.

Vibration data acquisition serves as an excellent baseline in the performance of the proposed sensor kit. When attempting to perform analytics on vibration data, it is important to determine a sensor which meets desired performance specifications for the intended application. As this work considers health monitoring of factory equipment such as motors, pumps, lathes and mills, a vibration sensor should be chosen which accurately captures signals in the frequency ranges at which these machines operate. For many machining and manufacturing applications, these machines operate at speeds at or below 150Hz. However, most defect frequencies for elements such as bearings manifest at several multiples of the fundamental spindle speed. For this reason, it is important to choose a vibration transducer with a bandwidth that exceeds 1.5kHz. In addition, this sensor must operate at or below 5 Volts to be compatible with the BeagleBone system.

Three accelerometers which meet the established criteria are shown in Table 5.2. First,

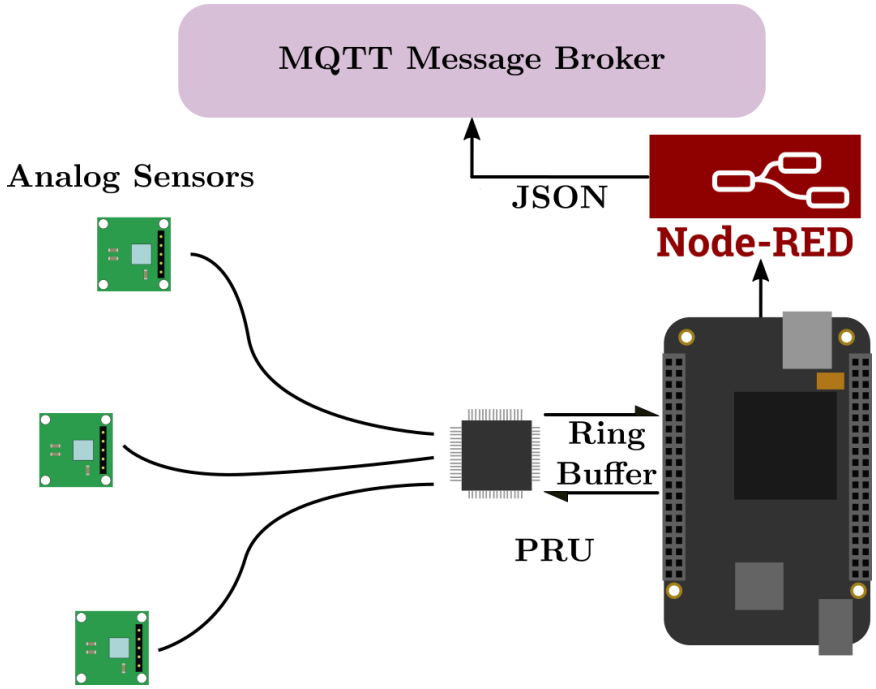


Figure 5.5: Schematic of analog sensor data acquisition

Table 5.2: Accelerometer Comparison

Name	**Price**	**Bandwidth**	**Sensitivity** (mV/g)
ADXL-1002	$36	11kHz	40*
ADXL-203	$19	2.5kHz	560
ADXL-335	$3	1.6kHz	300

* - Sensitivity at 5V supply

the ADXL-1002 is the most expensive sensor with the highest bandwidth at 11kHz. This large bandwidth comes at the expense of an anemic 40 mV/g sensitivity. Manufacturing equipment are generally designed to be well-balanced and exhibit low levels of vibration. For this reason, such a low sensitivity is unacceptable for the intended application. Between the AXDL-203 and ADXL-335, the the former has both better sensitivity and a larger bandwidth. The only downside of this accelerometer is its cost. However, the $16 price difference is considered acceptable for an increase of almost 100% in both bandwidth and sensitivity for the best possible data acquisition in a low-voltage system. For these reasons, the ADXL-203 is used in the proposed device.

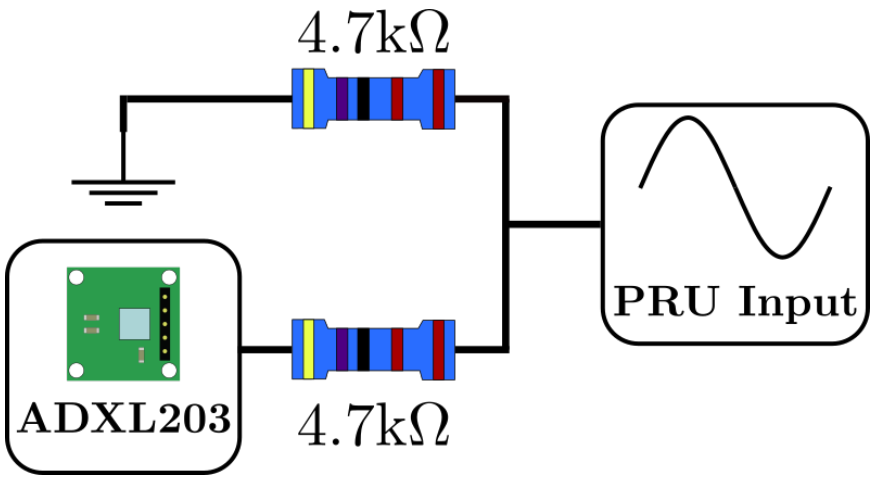


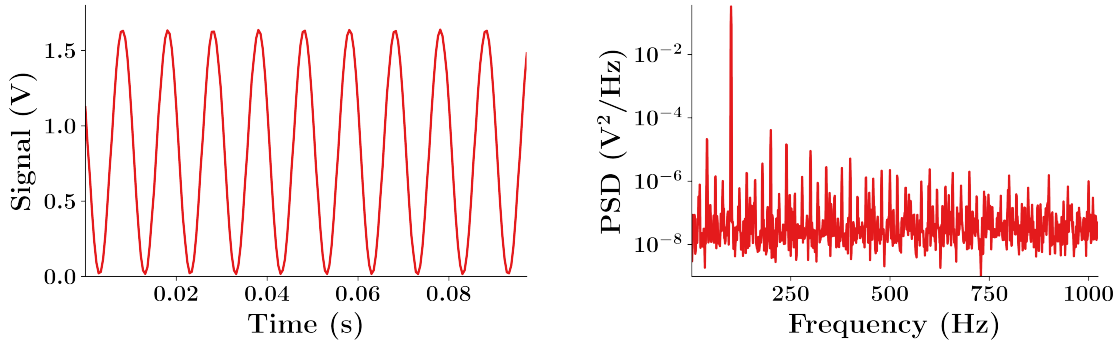
Figure 5.6: Analog voltage divider for BeagleBone ADC

In order to integrate the selected accelerometer into the BeagleBone-based data acquisition device, it is necessary to ensure that the signal voltage cannot exceed the maximum allowable level for the onboard ADC. As the BeagleBone Black and PocketBeagle have ADCs with 1.8V tolerance, the 3.3V signal must be divided approximately in half before being sent to the PRU. For this task, a simple voltage divider is applied to the signal from the accelerometer, as shown in Figure 5.6. Using $4.7\text{k}\Omega$ resistors ensures a peak voltage of 1.65V, provided a 3.3V supply to the ADXL-203. Notably, this voltage division also reduces the effective sensitivity of the accelerometer signal to $280\text{mV}/g$. This is still comparable to the ADXL-335 sensitivity without voltage regulation and is 700% better than the AXDL-1002 at 5V supply.

5.3.4 Data Acquisition Benchmarking

To assess the performance of the proposed data acquisition device, a simple sinusoidal signal is generated by an Arduino microcontroller. This signal is then captured by the BeagleBone device at various sampling rates to ensure consistency in the data acquisition process. As the Arduino device can produce a 3.3V analog signal, this test also ensures proper functionality of the voltage divider.

An example of this signal when sampled at 2,048Hz is shown in Figure 5.7. The



(a) Sampled signal

(b) Power Spectral Density curve

Figure 5.7: 2,048Hz sampling rate example

Table 5.3: Data Acquisition Performance Summary

Sampling Rate (Hz)	**Peak Voltage**	**Peak PSD**	**SNR (dB)**
2,048	1.638	0.326	28.3
4,096	1.645	0.326	31.3
8,192	1.645	0.325	34.4
16,384	1.644	0.324	37.4
32,768	1.651	0.324	40.4

sampled signal is shown in Figure 5.7a, which demonstrates an accurately captured sinusoid with a peak voltage of approximately 1.6V. This amplitude is almost exactly half of the true 3.3V output from the Arduino. This signal is captured very accurately in the frequency domain as well, as shown in Figure 5.7b. A very prominent peak, even on a logarithmic scale, is evident at exactly 100Hz in this plot.

A more thorough analysis of the signal sampling is shown in Table 5.3. As the sampling rate increases, the peak voltage more exactly reflects the 1.65V peak expected from a 3.3V source signal. In addition, the higher sampling rates increase the Signal-to-Noise Ratio (SNR), while the peak Power Spectral Density at 100Hz is approximately identical across all sampling rates. This result indicates that the BeagleBone data acquisition device can accurately capture analog signals at high sampling rates.

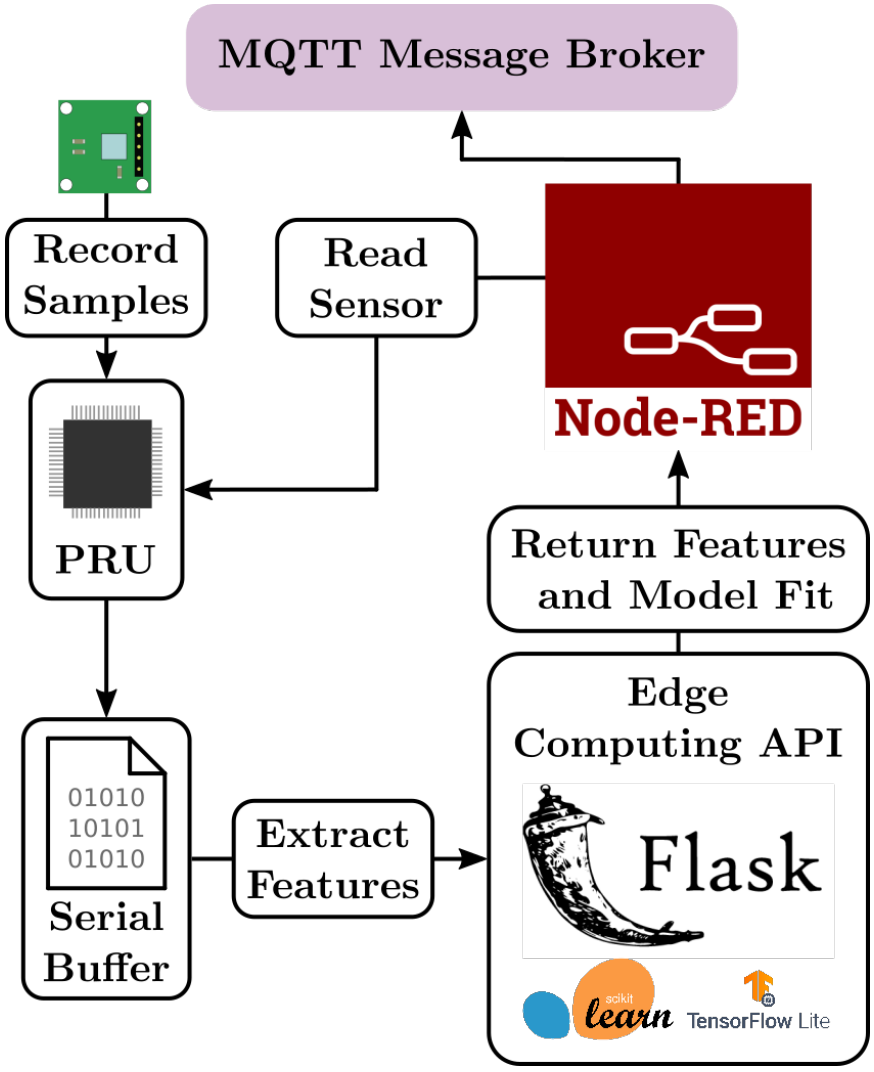


Figure 5.8: Functional diagram of the edge data acquisition device

5.4 Data Analytics

Using the BeagleBone platform for the proposed data acquisition device ensures access to all of the data analytics and machine learning tools available via Python and its associated libraries. Integrating data acquisition and statistical inference in one device ensures useful health monitoring data are made available directly from the edge. As a result, network traffic is reduced while also reducing latency in health monitoring applications.

A high-level functional overview of the data acquisition and model fitting process for the proposed edge device is shown in Figure 5.8. Central to the data acquisition process is

the Node Red server which handles incoming and outgoing data. A "Read Sensor" trigger is sent from Node Red to begin data acquisition from the PRUs. This trigger contains information such as sampling rate and number of samples to accommodate flexibility in the data acquisition process. The PRUs perform the analog-to-digital conversion on the sensor data, storing the 12-bit records in a raw serial buffer on the BeagleBone device. Once sampling is complete, the data are read into a Python API which extracts time and frequency-domain features on the data and performs model inference if a model is available. Once these computing tasks are complete, the data are returned to Node Red where they are formatted in a way that can be passed to the digital architecture via MQTT.

To integrate the proposed device in a near real-time health monitoring application, it must be capable of sampling, parsing, and fitting sensor data to a desired statistical model within a predictable time horizon. For this reason, a series of tests are performed to assess the efficiency of this device in each of these tasks. A sequence of one thousand samples are taken from the edge device. Timestamp information from Node Red are used to determine computational latency in sampling, feature extraction, and model inference for a varying number of vibration data points. An Amazon EC2 instance is used to compare the performance of the edge device with a more powerful computer for feature extraction and model inference.

5.4.1 Feature Extraction and Model Inference Performance

To evaluate the performance of the proposed device in vibration feature extraction and model inference, an array of time and frequency domain statistics are calculated from a sampled vibration signal. A representative sample of the extracted features is shown in Figure 5.9. The first four statistical moments are calculated along with the RMS to provide insight into the time-domain features. A Power Spectral Density estimation through Welch's Method is also calculated based on the size of the vibration sample. The number of points used to compute the DFT amplitudes is constrained such that the resulting

```
{
    "Kurtosis":-0.044158761565036286,
    "Mean":-0.37881984040141115,
    "RMS":0.019614941894183988,
    "Skewness":-0.011446712319675867,
    "Variance":0.00038475181635902136,
    "PSDAmps":[1.93e-7,1.24e-7,1.38e-7],
    "frequencyInterval":1
}
```


Figure 5.9: Extracted vibration features

PSD has a frequency resolution of 1Hz. For example, a 2,048-point DFT is used to compute the Power Spectral Density of a signal captured at 2,048Hz and so on.

The results of the signal processing performance test are shown in Figure 5.10. This plot shows the mean latency in parsing a sampled vibration signal with varying record lengths. Three separate conditions are presented in this bar chart. The first, "All Features," includes all time and frequency domain features as shown in Figure 5.9. The second, "Time-Domain," includes only time-domain features and excludes both the PSD computation and model fit. Finally, a "PSD Compute" condition shows the latency involved in computing all of the features in the first condition and performing the model fit, but only loading the time domain features back into Node Red. This condition is meant to highlight the latency involved in loading large arrays into Node Red.

As is clear from Figure 5.10, the Power Spectral Density estimation is a computationally expensive part of the feature extraction process. This figure shows the mean latency with error bars indicating the standard deviation. While increasing the number of sample points affects all three cases, computing and transmitting the PSD requires over one second for a vibration record length on the order of 65.5 thousand samples. Extracting time-series features alone from a similar sample requires less than 0.5 seconds.

Another interesting result of this test is the latency incurred by simply transmitting the

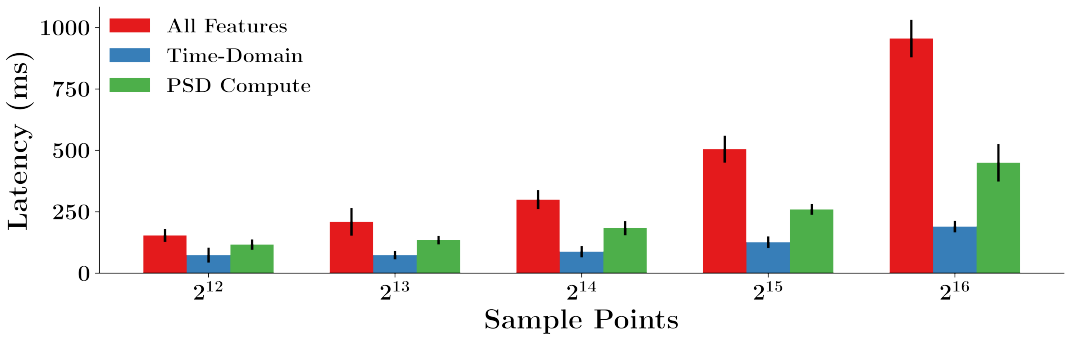


Figure 5.10: Feature Extraction Latency

PSD arrays into Node Red. This effect is especially true for larger array sizes. As the feature extraction is performed in Python due to its efficiency and ease in handling large arrays, this result indicates that better performance may be realized by eliminating Node Red from the vibration parsing process altogether. Still, this device derives high-quality data from vibration measurements and transmits these data within approximately one second of computation. For the vast majority of health monitoring applications, these performance metrics will suffice without additional optimization.

While computing the PSD of a vibration signal requires significantly more time than time-domain features alone, the results of this test indicate that the proposed device is capable of performing data acquisition and feature extraction on high-quality vibration data in a near real-time capacity. The latency percentiles of the smallest and largest samples are summarized in Table 5.4. Similarly to the previous plot, this table reveals how PSD computation requires substantially more time than time-domain feature extraction. Although the median latency for feature extraction without the PSD is approximately 300% higher for large samples than small samples, the 99^{th} percentile latency increases by only 30% for the same comparison. In either case, the maximum latency is on the order of 200ms. With this low level of latency, it is possible to track high-level statistics with precise timings. By comparison, the 99^{th} percentile latency for extracting frequency-domain information is 462ms for small samples and 1,274ms for large samples.

Table 5.4: Feature Extraction Latency Comparison

Record Length	**With PSD?**	**Median (ms)**	**95% (ms)**	**99% (ms)**
2^{12} (4,096)	Yes	172	204	462
2^{12} (4,096)	No	65	109	165
2^{16} (65,536)	Yes	922	1,075	1,274
2^{16} (65,536)	No	188	202	215

Although these latency numbers are significantly higher than without PSD calculations, the resulting features contain much more information and may be used to detect defects at specific frequencies. In either case, it is promising that high quality vibration features can be extracted with approximately 1 second of latency.

5.4.2 Comparison with Cloud Computing

While the feature extraction and model inference results appear promising, it is important to compare them with a cloud-based solution to assess the performance of edge computing. For this task, the vibration data are captured from the BeagleBone and published via MQTT to a publicly available message broker at *mqtt.eclipse.org*. An Amazon EC2 instance running a Python-based MQTT app subscribes to these data and performs both the feature extraction and model inference tasks before publishing the inference results back to the MQTT broker. The BeagleBone, subscribed to this MQTT topic, receives the inference results and determines the time elapsed since the data were published. This latency is compared to the total computation time required to perform the same feature extraction and inference tasks on the edge.

The results for this test are shown in Figure 5.11. This box plot shows the latency in processing one thousand vibration samples at each record length. The variance in the Amazon EC2 deployment is significantly higher than that of the BeagleBone. This is an expected result, given that the Amazon EC2 deployment requires transmitting a large payload to a cloud computer prior to computation. By comparison, the BeagleBone exhibits

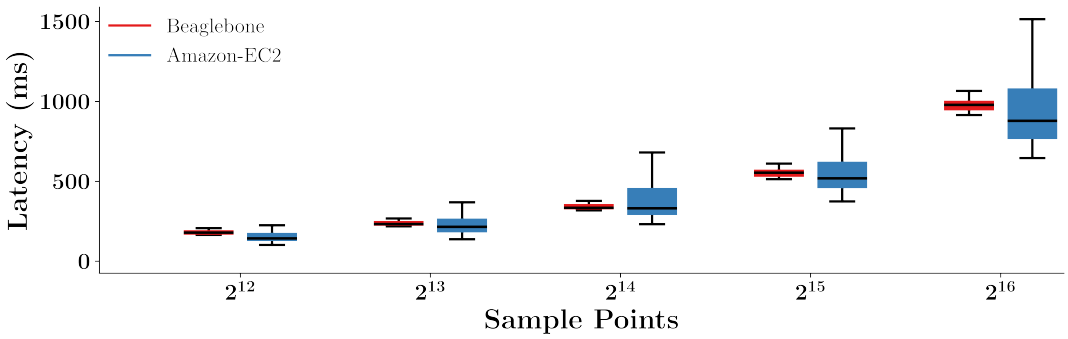


Figure 5.11: Total inference latency

Table 5.5: Edge vs Cloud Computing Comparison

Record Length	**Location**	**Median (ms)**	**95% (ms)**	**99% (ms)**
2^{12} (4,096)	Edge	180	206	261
2^{12} (4,096)	Cloud	143	893	4,262
2^{16} (65,536)	Edge	979	1,072	1,327
2^{16} (65,536)	Cloud	880	1,989	5,368

more consistent latency with a low variance. This low variation facilitates planning timing requirements for the data acquisition device.

Table 5.5 shows the latency from this test at different percentile levels and sample lengths. As indicated in the previous plot, the median latency values are relatively close between the Edge and Cloud deployment options. The median latency for the BeagleBone is approximately 10-25% higher than the Amazon EC2 instance. Looking towards worst-case latency, however, the BeagleBone appears to be much more reliable. While the 99^{th} percentile latency for the BeagleBone approaches 1.4 seconds for large samples, worst-case latency in the Cloud exceeds 5 seconds.

To determine the sources of transmission and processing latency for both the edge and cloud-based solutions, the latency involved in purely performing the computations are compared in Figure 5.12. This figure excludes all time involving data transmission. As a result, the Amazon EC2 instance shows much lower latency than the BeagleBone. While this is an expected outcome, it establishes that network latency and message transmission

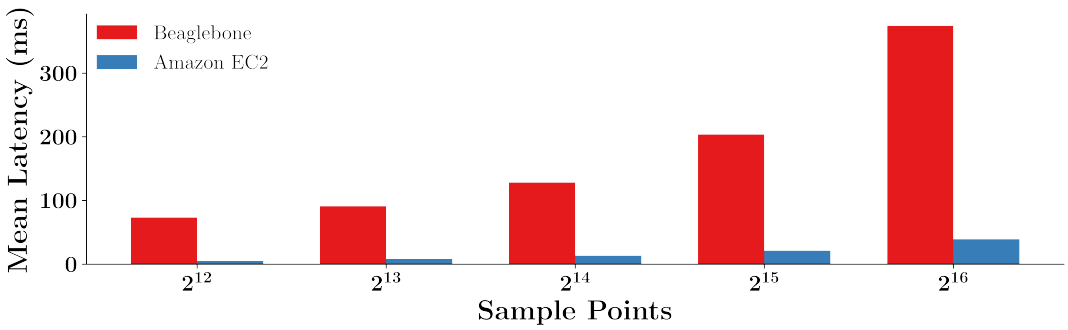


Figure 5.12: Local computational latency

are the primary drivers in determining the delay when processing sensor data in an IoT framework. Additionally, it is clear that larger payloads incur a higher latency penalty. This finding is supported by existing literature [98].

5.5 Conclusion

This chapter has introduced a proposed platform and methodology for performing CNC controller and analog sensor data acquisition in an IoT framework. Due to its unique real-time capabilities, the BeagleBone platform is proposed for this task. By leveraging Node Red, this device can accommodate multiple CNC communication protocols as well as analog sensor data. Furthermore, a strategy for performing feature extraction and model inference on the edge is proposed. Results show that this device is capable of capturing 64k-sample vibration arrays at sampling rates in excess of 30kHz. These data can then be processed for time and frequency domain statistics with computational latency on the order of one second. This computation time compares favorably against the latency involved in sending a vibration array to a remote cloud computer for feature extraction and inference.

CHAPTER 6
A CASE-STUDY IN TOOL WEAR

To fully explore an end-to-end application of the tools proposed in this thesis, a case study in tool wear is presented in this chapter. Specifically, this study investigates tool wear classification for end milling. A prototype version of the digital architecture is used to perform acquisition of machine controller and vibration data. An experimental design with a range of milling process parameters is replicated for both a healthy and unhealthy cutting tool to generate a varied data set. Analysis is performed on these data to determine which features are most predictive of tool wear. After choosing a promising model from the training data, it is validated through deployment to the edge device.

6.1 Background

To demonstrate an implementation of the proposed data acquisition and health monitoring strategy, a case study in tool wear monitoring is presented in this chapter. Specifically, this study investigates tool wear classification in an end milling process using the Emco E350 3-axis mill, pictured in Figure 6.1. This machine is equipped with a Siemens 828D controller and is capable of using the OPC-UA protocol for machine process monitoring. Spindle vibration signals are highly informative in assessing tool health. As this is an easily-injected fault condition in a healthy CNC machine, it is an excellent means of demonstrating the utility of the proposed health monitoring strategy.

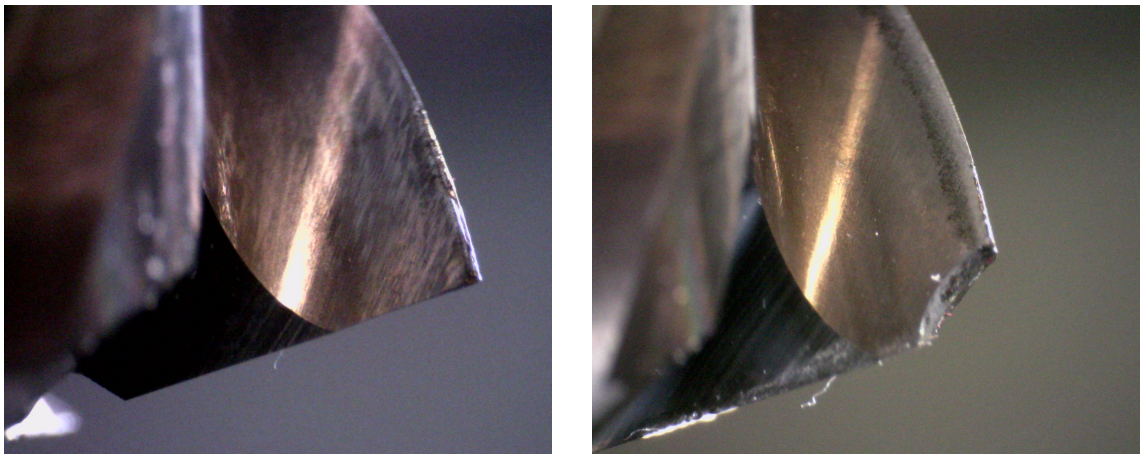
6.2 Methodology

Two milling tools are used for comparison in this case study: an unworn "healthy" tool and a badly chipped "unhealthy" tool. Both are shown in Figure 6.2. The healthy tool in Figure 6.2a is effectively unused, while the unhealthy tool in Figure 6.2b has large chips in

its leading edges. As these are both 4-tooth, 5/8-inch end mills, the images in these figures are typical of all four teeth. A robust classification model should be capable of detecting which tool is used in a slotting process, subject to variation in the cutting parameters.

To monitor the spindle vibration signal, an ADXL203 vibration transducer is mounted directly on the machine as shown in Figure 6.3a. In this image, the lid on the protective case has been removed to show the accelerometer setup. The accelerometer is mounted on standoffs normal to the spindle axis inside of a plastic box meant to help protect the electronics from coolant spray. This box is rigidly fastened to the spindle using hex screws. While it is optimal to fasten an accelerometer permanently with epoxy or a similar material for maximum vibration transmission, the less permanent fasteners are used here to allow machine operators to remove it if necessary for day-to-day operations. As this figure also shows, the Printed Circuit Board which houses the accelerometer is also coated in epoxy to insulate the electronics in the case of coolant ingress. A shielded M8 cable is

Figure 6.1: Emco E350 3-axis mill



(a) Healthy tool (b) Unhealthy tool

Figure 6.2: Tool quality comparison

connected to the plastic housing to provide power and route the vibration signal outside of the machine workspace, as shown in Figure 6.3b. This is done to protect the rest of the data acquisition computing hardware and minimize the number of additional components inside the machining area.

The accelerometer cable is routed to an enclosure containing the Beaglebone computer outside the Emco mill, as shown in Figure 6.3c. A voltage divider breakout board as described in Chapter 5 is mounted to the GPIO pins on the Beaglebone. An AC-DC power supply is also housed in this enclosure to convert the standard 120VAC wall outlet signal to a 5VDC signal which is safe for the Beaglebone. This enclosure and the Beaglebone hardware within it are capable of managing several analog inputs. For the purposes of this research, the single vibration signal is used.

6.2.1 Experimental Architecture

The digital framework used in this case study is summarized in Figure 6.4. An embedded Linux computer is connected to the Emco mill in order to retrieve OPC-UA controller data from the machine. A Node Red server running on this device formats the controller data in a way which is readily consumed by the MQTT-based digital framework. The

(a) ADXL-203 mount location

(b) Emco E350 mill interior

(c) Beaglebone Data Acquisition Device

Figure 6.3: Emco mill setup

Beaglebone-based data acquisition device is also connected to an MQTT message broker to send and receive messages. A remotely-located gateway subscribes to all of the controller and sensor data to populate a database for storage. These data are then used to train health

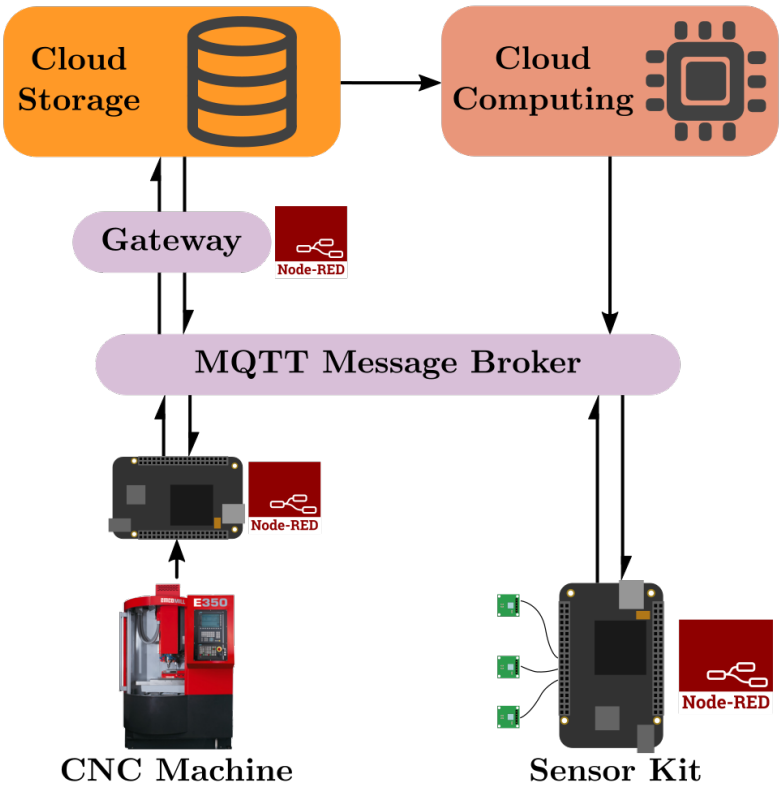


Figure 6.4: Experimental architecture

monitoring models. The most promising of which are sent back to the data acquisition device for real-time validation of the tool health monitoring approaches.

While this case study considers a single factory machine, the design of the IoT framework in this study is compatible with the proposed architecture as discussed in Chapter 3. MQTT messages sent from the vibration data acquisition device are labeled with the appropriate tags, as shown in Figure 6.5. The *assetId* of the machine is "EMCO-MMM," indicating the machine name and its location (Montgomery Machining Mall at Georgia Tech). The *sensorId* is "Spindle-Accel," indicating that it is monitoring accelerometer measurements on the machine spindle. To delineate between different cuts with predefined experimental parameters, the *iteminstanceId* field is used. Prior to performing a new cut at set feeds, speeds, and cutting depths, the data acquisition device is updated to reflect the current cutting condition. In this example, the value is "Sample-1." Beyond these specifics, the vibration payload contains all of the relevant statistics which

```
"topic": "Asset/EMCO-MMM/Vibration",
"payload": {
    "dateTime": "2020-03-08T15:43:44.808Z",
    "assetId" : "EMCO-MMM",
    "dataItemId" : "Vibration",
    "iteminstanceId" : "Sample-1",
    "sensorId": "Spindle-Accel",
    "samplingInterval":0.00005,
    "Kurtosis": 0.033140,
    "mean": -0.04032,
    "rootMeanSquare": 0.024866,
    "skewness": -0.204197,
    "variance": 0.000619,
    "PSDFreqInterval":4.883,
    "PSDAmps":[0.895,0.889,...,0.877]}
}
```


Figure 6.5: Labeled experimental vibration payload

have been discussed in previous chapters. Notably, the chosen sampling rate is 20kHz to avoid aliasing. With this sampling rate, a 4096-point FFT is used for the Power Spectral Density approximation, resulting in a spectrum resolution of 4.88Hz.

As a test of the data acquisition device and classification model robustness, the Arduino-based sampling platform described in Appendix A is used in the model training phase. As the ADC on this device is 3.3V tolerant, the Arduino-based DAQ is approximately twice as sensitive as the Beaglebone-based DAQ. This benefit comes at the cost of significantly reduced memory availability and overall data throughput. As this data acquisition device is not well-suited for near real-time statistical inference, the modified, Beaglebone-based system is used to validate the classification models.

In this case study, the controller data are used to help label the accelerometer data. The process flow for this labeling is shown in Figure 6.6. The data acquisition device subscribes to OPC-UA parameters such as spindle speed and program name. While the spindle is powered off, no interesting process or health information can be extracted from the spindle

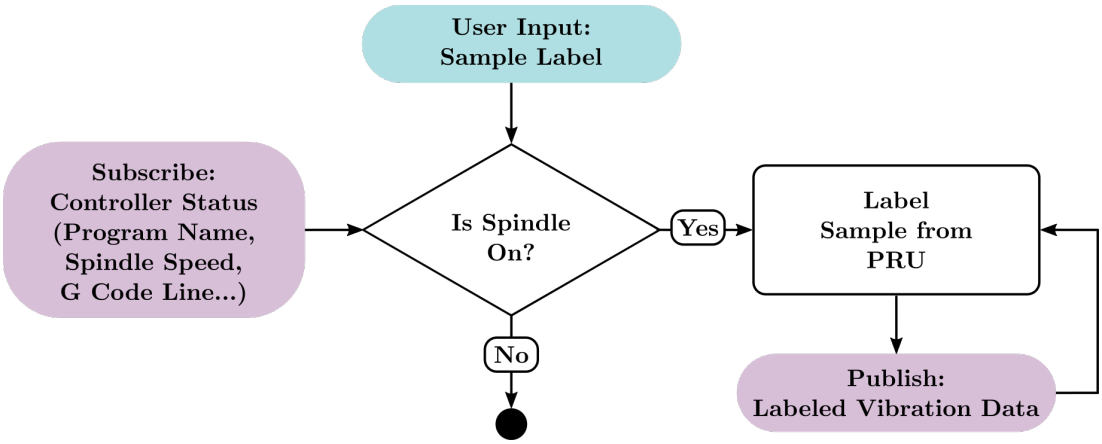


Figure 6.6: Vibration labeling

Table 6.1: Experimental Parameter Levels

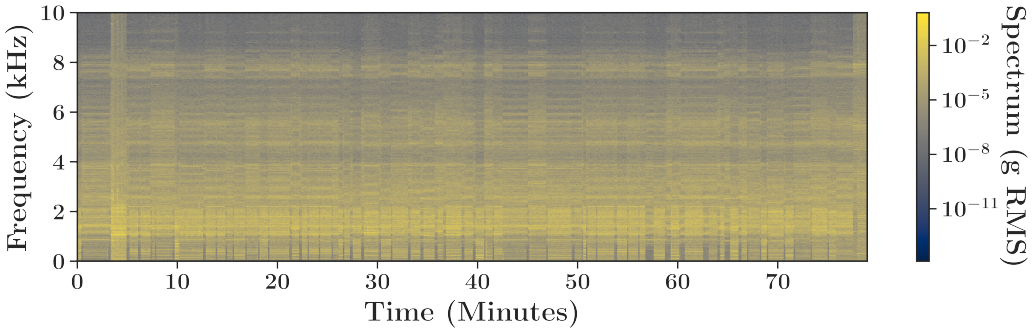
Parameter	Low Level	Mid Level	High Level	Units
Surface Speed	200	250	300	ft/min
Chip Load	0.001	0.002	0.003	in/tooth
Depth of Cut	0.0313	0.0938	0.1563	in

accelerometer, so no vibration message is sent to the message broker. Otherwise, the data acquisition device creates the JSON payload and publishes the vibration data. This ensures minimal extraneous vibration data are sent to the database for model training.

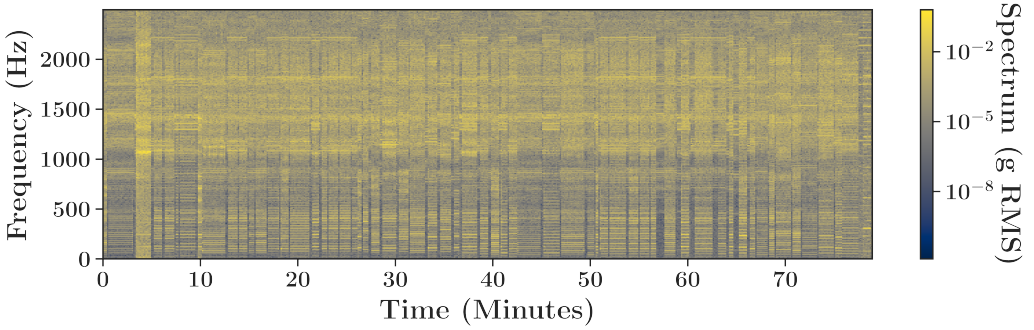
6.3 Model Training

To train a robust classifier, a varied, balanced training data set must be captured. The process parameters must reflect typical ranges used when machining the material of choice. For this study, mild steel is used. The resulting feeds, speeds, and cutting depths are shown in Table 5.4. These parameters are chosen to keep spindle motor utilization below approximately 25% of its rated power. As the Emco E350 is not an exceptionally powerful machine, these relatively conservative values are used.

With an established parameter space, a training experiment is generated by fractional factorial design with centering. A total of 20 training sample conditions are created in



(a) Full spectrogram



(b) Low-frequency spectrogram

Figure 6.7: Experiment spectrograms

this manner. Similarly, a testing experiment is generated by using programmatic random number generation to select values within the predefined limits. Ten total sample conditions comprise the testing set. These training and testing sets are randomized and replicated for both the healthy and unhealthy tool. In total, 60 experimental conditions are used for the training and testing data sets in this study.

6.3.1 Data Analysis and Processing

After performing the experimental cuts, a total of 560 vibration samples are captured over the course of approximately 80 minutes. The full spectrogram of these vibration data is shown in Figure 6.7a. From this plot, different segments of steady-state vibration are apparent. These distinct periods of time indicate different experiment samples with varying process parameters. While the Nyquist frequency of the sampled signal is 10kHz, the

majority of the signal energy is concentrated at frequencies below approximately 2kHz. The nuanced changes in energy content at these low frequencies are more clearly evident in Figure 6.7b. In this spectrogram, two distinct segments are evident. At low frequencies below approximately 500Hz, discrete bands appear. These frequency components relate to the tooth-pass frequency and harmonics for different cutting conditions. At frequencies between approximately 1kHz and 2kHz, broadband noise, due in part to machine dynamics and tooth chatter, are apparent. These high-frequency components are also important in determining tooth health as worn tools generally exhibit higher energy in this frequency range.

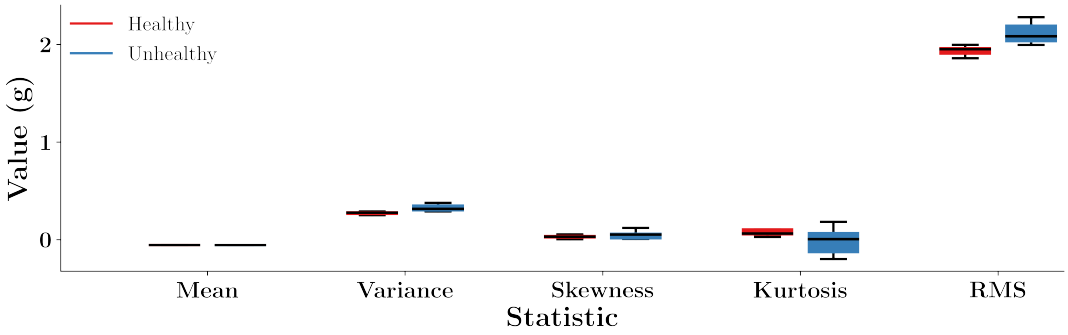
From this full dataset, processing is done to prune and organize it into training, testing, healthy and unhealthy data. For each sample, vibration data are captured prior to tool engagement and after the tool has exited the material. These vibration data are excluded from this analysis as they are not reflective of tool quality. While they could be used to create a classifier to determine whether the milling machine is cutting material or not, this task is outside of the scope of this chapter. After manually pruning these samples, 457 vibration payloads remain which represent active machining operations. These samples are broken down as shown in Table 6.2. As intended, 67% of the data represent the training set. This subset is well-balanced, with a 0.2% margin between healthy and unhealthy samples. The remaining 33% of the data are reserved for testing. This data set is less well-balanced with a 13.8% margin between unhealthy and healthy samples. This discrepancy between healthy and unhealthy data is innate in the randomization of process parameters for the testing samples. With ten randomly chosen combinations for both the healthy and unhealthy condition, the unhealthy parameters are slightly slower. As a result, they require slightly longer to complete compared to the healthy parameters, generating more data samples in the process. In all, the breakdown of this dataset is promisingly balanced. Based on this composition, an unbiased classifier can feasibly be created from the training data.

Table 6.2: Data Breakdown

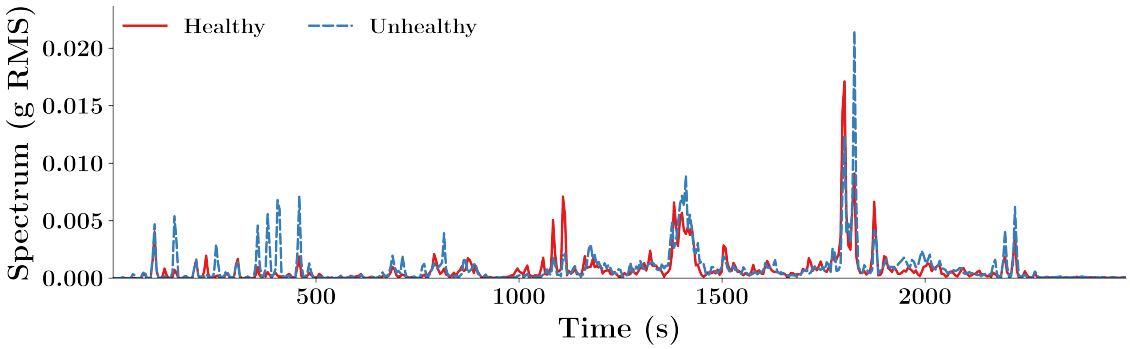
Category	Total Samples	Percent Healthy	Percent Unhealthy
Train	306	50.1%	49.9%
Test	151	43.1%	56.9%

To highlight distinctions in sample data, the vibration features from a healthy and unhealthy sample are shown in Figure 6.8. The process parameters for both samples are identical apart from tool health. The distribution of the time-domain statistics is shown in Figure 6.8a. From this figure, it is clear that the mean is stationary, as would be expected from a steady-state signal. Small shifts in the Variance, Kurtosis, and RMS values are apparent from the healthy and unhealthy signals, while the Skewness is not obviously shifted. Based on this comparison, it may be possible to use these time-domain statistics to assign a tool health classification for this simple example. By comparison, the signal spectrum reveals some interesting differences between the healthy and unhealthy tool. Again, at low frequencies below 500Hz, the unhealthy tool exhibits significantly higher levels of vibration at the tooth-pass frequency and its harmonics. In addition, a sharp peak is apparent at approximately 1,800Hz in the unhealthy signal.

While the time-domain statistics are efficiently computed and highly predictive in the simple case where the experimental parameters are identical, their predictive power is limited in an experimental design with high variability in the intensity and nature of vibrations. This fact is illustrated in Figure 6.9. Here the time domain statistics for the full training dataset are compared for the healthy and unhealthy tool conditions. Clearly, the chosen statistics are insufficiently separated to yield any meaningful conclusion from them. For all five statistical metrics, their distributions are nearly identical. For this reason, the spectrum features are considered baseline for this analysis, instead of the time-domain statistics.



(a) Time-domain statistics comparison



(b) Spectrum comparison

Figure 6.8: Healthy vs Unhealthy data - 250SFM, 0.002IPT, 0.1563” Depth of Cut

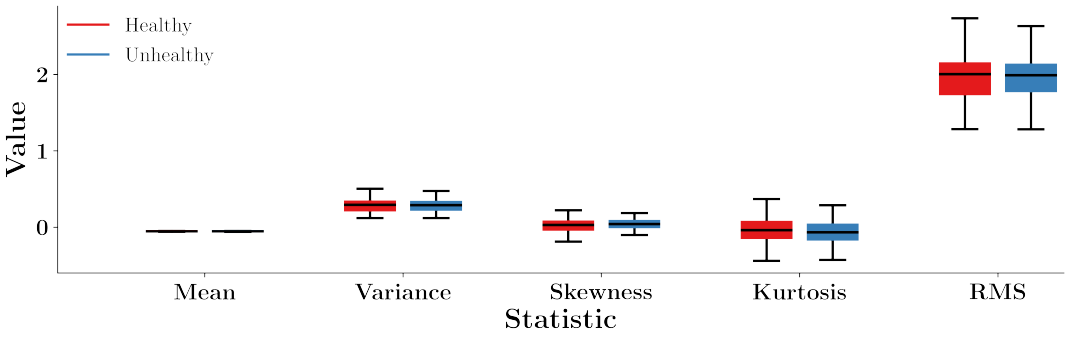


Figure 6.9: Full time-domain statistics comparison

6.3.2 Naive Bayes Model

To assess the quality of various model choices, a PCA-GNB model is used as a baseline. This selection is made for a number of reasons. First, the Naive Bayes classifier is deterministically trained. As a probabilistic classifier, it is not sensitive to randomization

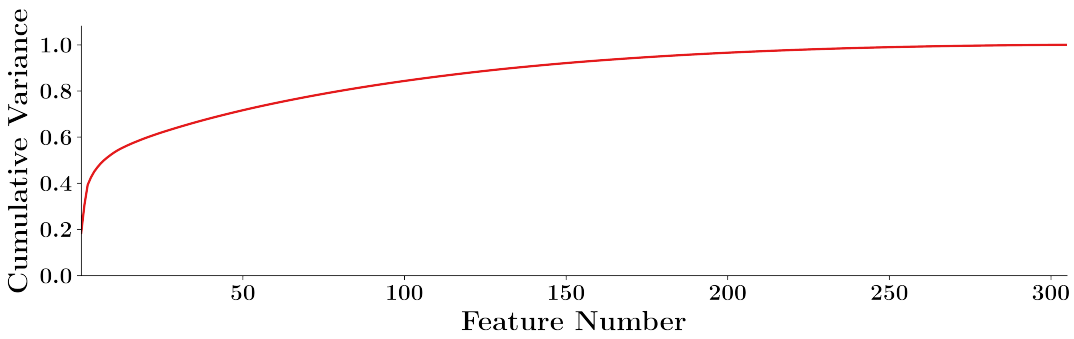


Figure 6.10: PCA decomposition of the spectrum features

in the same way that neural network-based classifiers are. The Gaussian Naive Bayes classifier also has very few hyperparameters. As a result, it is easy and efficient to evaluate various hyperparameter choices and visualize their influence on the model outcomes. This assists in the explainability of the model. Ultimately, using such a model with readily explained underlying statistics improves confidence in its conclusions.

The cumulative explained variance of the training data is shown in Figure 6.10. This plot shows that the first several features are highly influential in capturing the variance in the dataset, while increasing the number of features past 10 or so gradually increases the explained variance. When down-selecting the features for model development, a balance must be sought between using too few features which do not adequately represent the full dataset and using too many features which cause the model to fit to noise.

Beyond determining the appropriate number of features to use from PCA decomposition, it is important to establish which features to use from the full set of available vibration statistics. Three groups of features are investigated in this study. First, the frequency-domain spectrum features are considered. This group of features is by far the largest and contains detailed information about the prominence of various frequencies in the vibration signal. Next, the time-domain statistics provide high-level information about the signal as a whole. While they do not contain as much information as the spectrum features, they may provide important nuance to these features. Finally, contextual information is

available from the machine controller. Information such as feed rate and spindle speed are readily extracted form the OPC-UA server and can be used to improve model performance when integrated with the other vibration features.

To evaluate the performance of classifier models with various feature combinations, a large number of PCA-GNB models are trained to classify the nominal and anomalous tool state with varying feature inputs and PCA dimensionality. These models are then evaluated on the testing data to determine how well they generalize from the training dataset. With these performance metrics, an optimal hyperparameter combination is chosen to maximize an objective function

$$J = F_{train}^2 + F_{test}^2 \tag{6.1}$$

where F_{train} and F_{test} are the F1 scores of the training and testing data, respectively. These scores, derived by computing the harmonic mean of the model precision and recall, provide a balanced metric for classifier performance.

A grid search is used to vary the PCA dimensionality and spectrum components for the classifier. Because the bandwidth of the ADXL203 accelerometer is 2.5kHz, only spectrum components at or below this frequency are considered for the model. This reduces the maximum dimensionality of the 20kHz sampled signal by 75%. These spectrum features are combined with chosen time-domain statistics or OPC-UA labels before being reduced in dimensionality by PCA. Because the cumulative explained variance of the training spectrum data exceeds 90% at approximately 128 features, this is used as the upper bound for PCA dimensionality.

A representative plot which summarizes classifier performance as a function of PCA dimensionality and spectrum bandwidth is shown in Figure 6.11. This figure shows the normalized objective function value for PCA-GNB models trained with spectrum features only. Two interesting points are made clear from this plot. First, the very low frequencies on the order of 100Hz have meaningful predictive power. These frequencies correspond to the tooth-pass frequencies of the tool. By using a very small number of features from these

low frequencies, a simple, relatively accurate classifier could be created. Second, there is a very large band between approximately 100Hz and 1,000Hz where adding spectrum bandwidth to the classifier does not improve model performance. This is followed by a spike in performance when higher frequencies are used. Based on previous literature, this result makes sense; these high frequencies at approximately 1,000-1,500Hz are associated with tool chatter dynamics, and are generally more prominent in tools with high wear. The trends visible in this plot instill confidence that the GNB model learns to generalize from the available features to correctly identify the frequencies which are truly related to tool wear.

The full comparison of varying feature combinations and model performance metrics is shown in Table 6.3. Four feature combinations are shown which include the spectrum, time-domain statistics, and OPC-UA labels. The optimal PCA dimensionality, spectrum bandwidth, and testing scores are shown for each combination. Interestingly, the model which uses the spectrum features alone exhibits the best overall performance. In addition, this model is the smallest, with 31 PCA dimensions and a spectrum bandwidth of 1,470Hz. When the OPC-UA labels are added to these features, the testing precision and recall scores are approximately the same with an increase in 5 dimensions from PCA. From this table, it is clear that including the time-domain statistics more than doubles the PCA dimensions and significantly hampers testing performance. This result is expected based on the poor

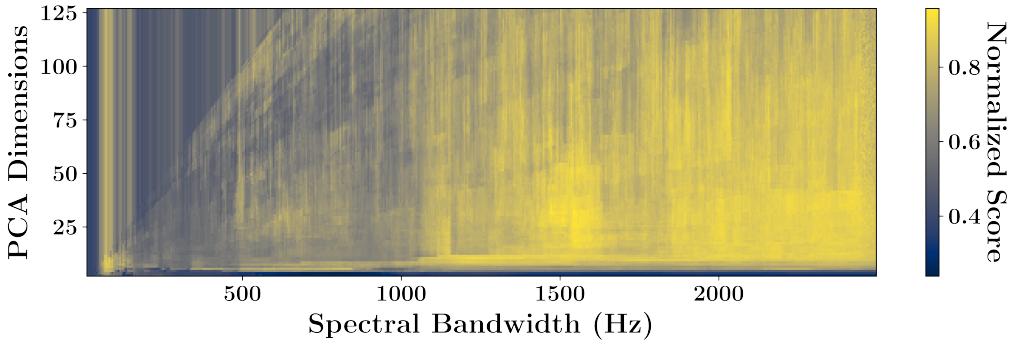


Figure 6.11: Classifier performance with varying hyperparameters

Table 6.3: Feature Performance Comparison

Feature Combination	Optimal PCA Dimensionality	Optimal Spectrum Bandwidth	Test Precision	Test Recall
Spectrum Only	31	1,470Hz	96.5%	94.8%
With OPC-UA	36	2,237Hz	94.8%	94.8%
With Stats	76	410Hz	80.7%	85.2%
With OPC-UA and Stats	121	1,158Hz	84.2%	78.7%

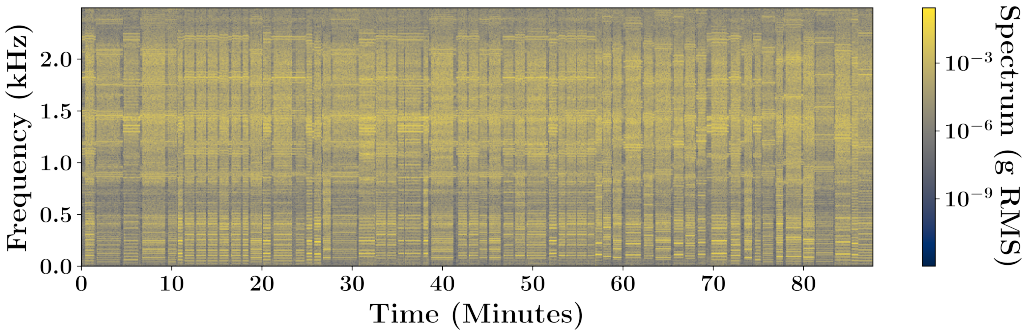


Figure 6.12: Spectrogram of full validation test

separation of these statistics with varying process parameters.

6.4 Model Validation

To validate the best-performing model, it is deployed to the Beaglebone-based data acquisition device for real-time inference. Because the model is trained on the Arduino-based, 3.3V tolerant accelerometer data and is therefore more sensitive to changes in acceleration, the model is expected to experience some performance degradation when deployed to the Beaglebone. In addition, due to closures related to the COVID-19 pandemic, this validation takes place after approximately 3 months of machine downtime. While each of these factors individually should have minimal impact on the captured vibration signal from the spindle, they introduce small variations from the training data which may be used to determine the robustness of the classification model.

To validate the chosen model, identical process parameters are used from the training

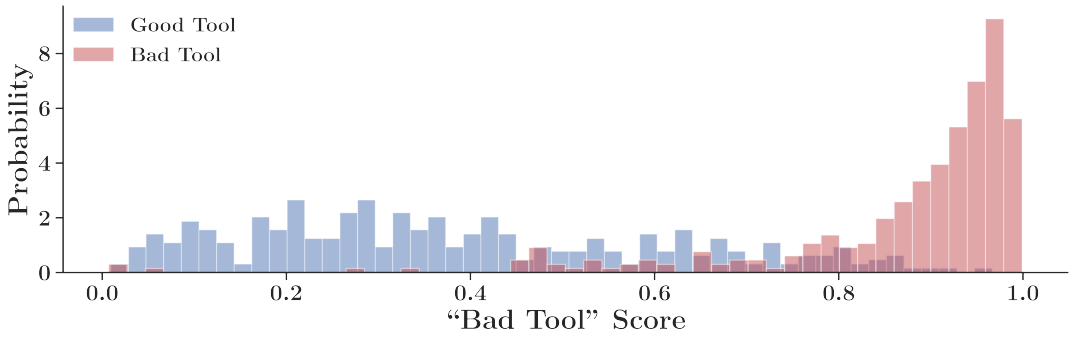


Figure 6.13: Classifier output for validation data

and testing data. Again, the frequency and time-domain statistics are extracted from the data generated in this validation step. After these statistics are captured from the vibration data, the relevant spectrum features are passed to the PCA-GNB model onboard the Beaglebone. This model generates a number between 0 and 1 indicating the predicted health of the tool. The full spectrogram of this validation dataset is shown in Figure 6.12. Again, these data cover approximately 90 minutes of machining time. Importantly, these spectrum features are shown as parsed onboard the edge data acquisition device. Using the low-latency processing capabilities of the Beaglebone, these frequency-domain features are extracted in real-time and sent to the digital architecture.

After processing the vibration features, they are passed to the onboard PCA-GNB model for inference. A total of 335 and 332 vibration samples are captured from the healthy and unhealthy tools, respectively. These samples are captured at an interval of once per three seconds. The classifier output from the full dataset is shown in Figure 6.13. From this plot, it is clear that the classifier is much less confident in labeling the healthy tool as compared to the unhealthy tool. For a typical classification implementation, a sample is classified as "healthy" if the model output is less than 0.5 in this scale, and "unhealthy" if the model output is greater than 0.5. By this metric, the performance of the model is summarized in Table 6.4. While the true positive rate, which indicates the success of the model in detecting an unhealthy tool, is 94.6%, the true negative rate is significantly lower

Table 6.4: Validation Performance Summary

Balanced Accuracy	True Positive Rate	True Negative Rate	Precision	Recall
85.4%	94.6%	76.2%	0.700	0.946

at 76.2%. Importantly, this model very effectively detects when the unhealthy tool is used, and can be used to flag potential problems in the machining process.

6.4.1 Control Chart Analysis

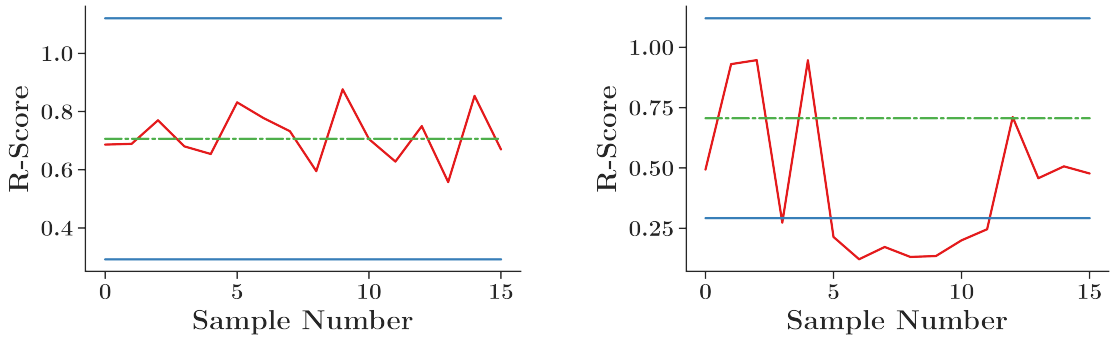
To further assess the performance of the PCA-GNB model, statistical control charts can be used. Using the classifier output as the process parameter to track, the mean and range of the output are used to generate $\bar{X}$ and R charts. These statistics are taken from the one-minute rolling average of the model output. As the data acquisition rate is 1 sample per 3 seconds, a 20-sample mean and range is used in computing these charts. The first 25% of the data from the good tool are used to construct the limits of the $\bar{X}$ and R charts. For an R-chart, these control limits are given by

$$UCL = D_4 r \tag{6.2}$$

$$CL = \bar{r} \tag{6.3}$$

$$LCL = D_3 r \tag{6.4}$$

where D_3 and D_4 are constants dependent on the number of samples used in the mean calculation. In the 20-sample case, $D_3 = 0.414$ and $D_4 = 1.586$. After computing these control limits, the R-Charts for the good and bad tools are shown in Figure 6.14. As Figure 6.14a shows, the variation in the model output for the good tool is consistent and within the control limits. By comparison, Figure 6.14b reveals that the model provides uncharacteristically low variation in classifying the bad tool. This information can be used to flag potential outliers in the manufacturing process.



(a) Good Tool

(b) Bad Tool

Figure 6.14: R-Charts for validation data

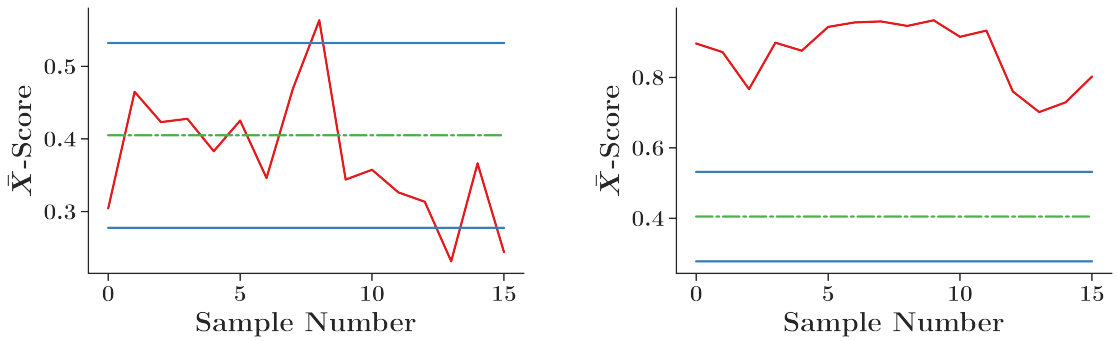
Similarly, an $\bar{X}$-Chart can be generated by analyzing the mean of the model output. The control limits of an $\bar{X}$-Chart are defined by

$$UCL = x + A_2 r \tag{6.5}$$

$$CL = x \tag{6.6}$$

$$LCL = x - A_2 r \tag{6.7}$$

where $A_2 = 0.180$ is a constant based on the number of samples in each sample mean. When these control limits are computed for the good tool data, the full control charts are as shown in Figure 6.15. The results for the good tool are shown in Figure 6.15a. Due to the large variance in the data, some points fall outside of the control limits. Because the ground truth is known in this case, the out-of-control data points reveal that the classifier model performs exceptionally poorly at certain process parameters. This information may be used to further optimize the model to ensure better overall performance. By comparison, the $\bar{X}$-chart for the bad tool is shown in Figure 6.15b. From this plot, it is abundantly clear that the mean classifier output is substantially outside of the control limits. This result follows from the fact that the classifier consistently produces confident results when the tool used is actually of poor quality.



(a) Good Tool

(b) Bad Tool

Figure 6.15: $\bar{X}$-Charts for validation data

6.5 Conclusion

This chapter has presented a case study in tool wear monitoring with the data acquisition and processing strategy proposed in this thesis. By varying the manufacturing process parameters, a robust classifier is created which accurately identifies when an excessively worn tool is used. By using a simple, Gaussian Naive Bayes classifier with Principal Components analysis for the dimensionality reduction step, model performance is easily compared for a wide variety of hyperparameters. This model is then deployed on the proposed data acquisition platform and implemented in real-time. The validation experiments demonstrate the robustness and ease with which this model may be implemented with the proposed strategy. By introducing control charts, a more robust explanation for model performance may be generated to track tool health over time.

CHAPTER 7
CONCLUSION

This thesis has presented an integrated approach towards machine health monitoring in a manufacturing environment. Utilizing the Internet of Things, machine controller data were captured along with external sensor data to provide a full picture of the machine operational status and health metrics. Using a standardized messaging structure, data streams from both sources were readily recorded in cloud databases. These data were then queried to facilitate model creation which may be used to track the health of the machine over time.

Leveraging state-of-the-art machine learning and statistical libraries, a large array of models with varying complexity were deployed to the edge for near real-time inference. In doing so, network latency was reduced. By integrating the model inference onboard the data acquisition computers, a compact, low-cost solution for IoT health monitoring was created.

This full proposed framework was demonstrated in a tool health monitoring application on a 3-axis mill. After generating a varied set of experimental parameters, the data acquisition tools were deployed to the Emco E350 mill. By recording both the controller and accelerometer signals from the machine, a statistical model was readily created to predict whether an unworn or excessively worn end mill was used. This model was then deployed to the data acquisition device and run in real-time to validate its performance.

These contributions are summarized below:

1. Development of a digital architecture which facilitates contextual edge analytics – Chapter 3

   In this chapter, a strategy for integrating controller and sensor measurements in an IoT framework was proposed. Basic demonstrations on a 3-axis mill were

used to highlight long-term health monitoring applications.

2. Edge-deployable anomaly detection and classification algorithms – Chapter 4

   This chapter summarized the deployment of state-of-the-art machine learning toolboxes and models to an embedded Linux computer and compares model performance across multiple alternatives. The results from this chapter revealed that even moderately complex neural networks may successfully be deployed to these edge devices with near real-time inference.

3. An edge device capable of controller and sensor data acquisition and analytics – Chapter 5

   This chapter described the design and implementation of an embedded Linux computer capable of high-quality sensor measurements, model inference, and IoT connectivity. This device was benchmarked against state-of-the-art cloud computing capabilities to compare worst-case message transmission latency.

4. An example deployment of this architecture and edge device for tool health monitoring in a 3-axis mill – Chapter 6

   Leveraging all of the methods proposed in the preceding chapters, a case study was undertaken in this chapter to investigate the efficacy of the proposed strategy in real-time health monitoring. Specifically, a statistical classifier was trained to detect when an excessively worn tool is used in an end-milling process.

7.1 Future Work

Given the breadth of subject matter addressed in this thesis, ample work remains to be done towards the development of advanced data acquisition in machine monitoring applications.

First, further investigations remain to be undertaken with regards to sensor data acquisition within this framework. While this thesis demonstrated the use of high-density accelerometer data, other data streams may remain problematic and necessary in the health monitoring of manufacturing processes. For instance, a similar approach may be taken to transmit and store image data from a factory floor. While accelerometer data is generally high-volume, high-resolution images are even more so, creating a potential opportunity to develop strategies for these data streams.

While this thesis proposed a framework for contextualizing sensor data within an IoT architecture, it is especially important to track the data streams resulting in fabricating individual parts for the purpose of qualification. Further work may illuminate methods of efficiently and automatically tracking the data streams of individual parts as they undergo various manufacturing processes. In doing so, it may become increasingly possible to deliver "born-qualified" components, with a robust and detailed record of data generated from its manufacture.

At the time of this writing, significant advancements continue to be made regarding embedded machine learning inference on ultra low-power devices. While the Beaglebone and embedded Linux platforms are excellent low-cost computers, they cannot effectively be run on battery power for long periods of time. This limitation may inhibit the utilization of the proposed strategy in factories where power is not readily available. Future work may be used to investigate the use of increasingly powerful microcontrollers and their open-source software which may enable true, real-time machine learning inference.

Appendices

APPENDIX A
ARDUINO-BASED DATA ACQUISITION DEVICE

A.1 IoT Enabled Sensor Pack

While modern CNC machines possess significant built-in functionality for machine state monitoring, many operations-critical assets and legacy manufacturing equipment do not have such capabilities. When such devices must be monitored, a standardized, modular sensor device is useful. With recent advancement in low-cost, low-power embedded devices, it is now feasible to develop such a platform in an economically sensible way.

A.1.1 Hardware Design

Given all of the aforementioned technological advancements, Georgia Tech has developed a standardized package for integrating sensors with the proposed digital architecture. The base of this kit contains two main components: the BeagleBone Black and the Teensy microcontroller. These components integrate with any standard sensor which uses analog or common digital communications protocols such as I^2C and SPI. M8 connectors are used to plug the sensors into the enclosure for easy data transmission.

This sensor pack is pictured in Figure A.1. In this example, an accelerometer is plugged into the Analog-to-Digital (ADC) conversion pins of the Teensy microcontroller. The converted digital signal is transmitted to the Beaglebone Black via the synchronous serial protocol, UART. Because the ADC must operate on the order of hundreds to thousands of samples per second for each sensor, this real-time task is reserved for the embedded microcontroller. The Beaglebone Black, an embedded Linux device, possesses networking capabilities and advanced computational libraries which microcontrollers typically lack. For example, a machine health classification algorithm built on well-supported Python

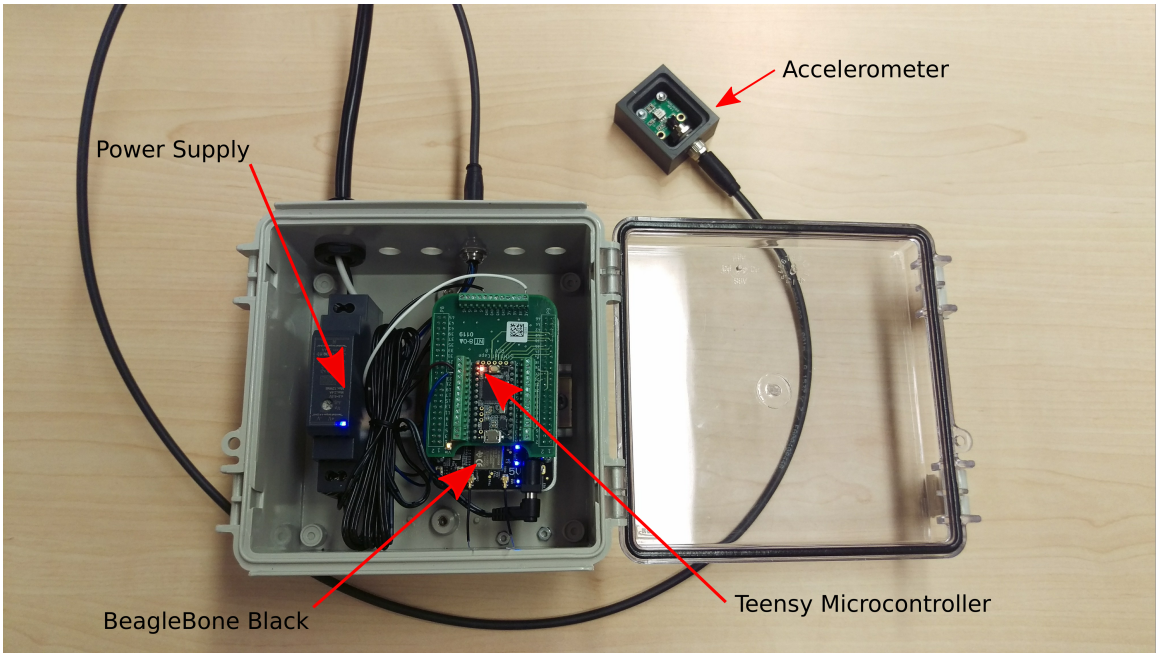


Figure A.1: IoT-Enabled Modular Sensor Platform

libraries could be deployed on the Beaglebone Black as an edge computing device. By engineering the IoT sensor pack in this way, a user can ensure properly-timed, deterministic sensor sampling which does not interfere with the other computational requirements of the edge device.

By leveraging the Linux operating system and its associated libraries, an engineer can rapidly prototype the network communication tasks involved in the digital architecture in a way that can be readily transferred to any other Linux system such as the Raspberry Pi or Nvidia Jetson Nano. This flexibility engenders scalability; as Linux systems continue to gain popularity in the IoT community and naturally increase in computational power, this platform will serve as a solid foundation on which increasingly demanding machine health monitoring applications can be developed.

The main functional components of the IoT sensor pack are illustrated in Figure A.2. An array of analog sensors may be attached to the ADC pins on the Teensy microcontroller. This embedded computer performs real-time measurements, storing arrays of measured voltage. These arrays are limited to sizes which can be stored in the onboard UART buffer;

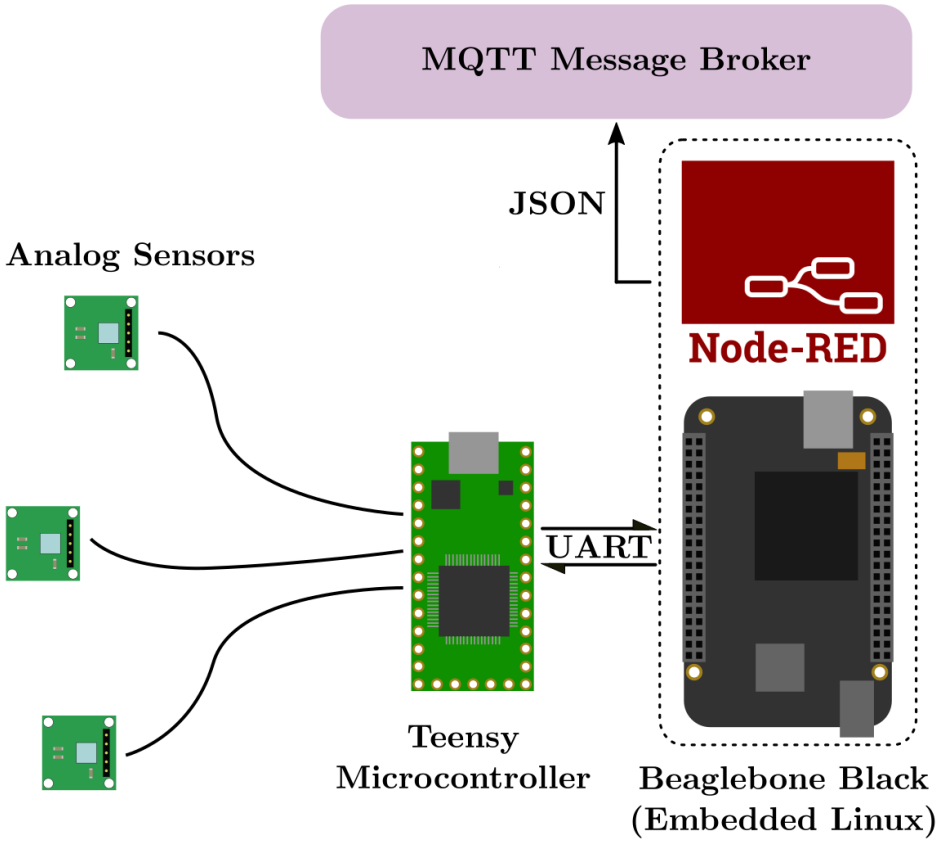


Figure A.2: Diagram of the primary sensor pack components

for the Teensy specifically, this limit is approximately 4096 bytes. At predefined intervals, the sensor measurements are transmitted to the Beaglebone Black via UART. Importantly, this serial connection accommodates bidirectional information flow. Thus, parameters such as sampling rate and array size may be specified from the Linux device and transmitted to the microcontroller. As a result, the time interval between samples in the array of received data is known ahead of time. This information is used to properly assign timestamps to each data point.

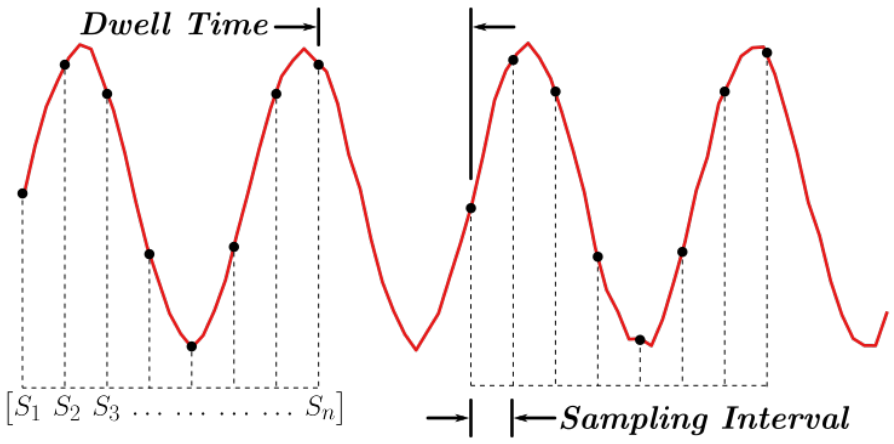


Figure A.3: Demonstration of accelerometer sampling for the sensor pack

A.1.2 Microcontroller Software

The sensor pack is designed to sample accelerometer data in a manner described by Figure A.3. The limited memory and buffer size of the Teensy microcontroller mandate that data be sampled in incremental bursts. These data are deterministically sampled at a rate defined by the *Sampling Interval* parameter. A total of N samples are captured for every sample array. The *Dwell Time* parameter dictates the amount of time between sample arrays. The Teensy microcontroller is programmed to accept updates to each of these parameters via UART string commands while it is not sampling. For example, the following command sets the accelerometer sampling rate to 10kHz and the number of samples, N, to 1024:

```
SetAccelerometer|10000|1024
```

Similarly, the *Dwell Time* can be updated by the following input:

```
SetInterval|0.017
```

which is given in units of Hertz, and roughly corresponds to one minute between samples.

If the array of data exceeds the UART buffer size, it is split into multiple UART strings. The sensor type is prepended to each of these strings, and an "END" string is appended to the final string. For example, if the Teensy is sampling from two accelerometers and

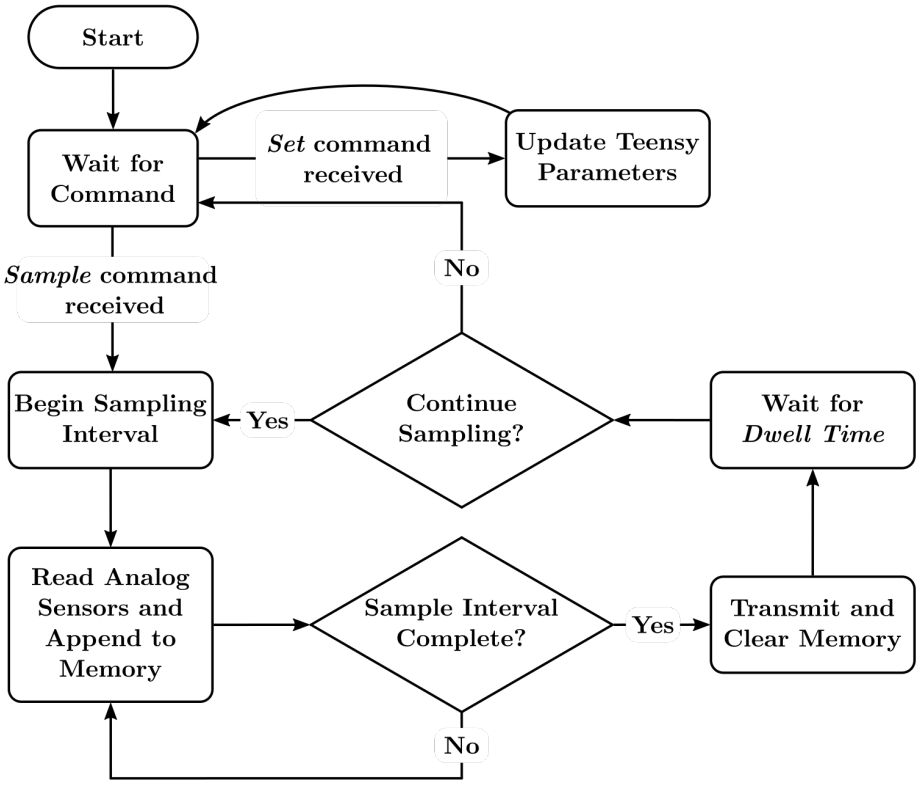


Figure A.4: Microcontroller process flow diagram

the length of the array exceeds the buffer size, the UART strings will be transmitted in the following way:

```
Accelerometer-1,2.071, 2.071, ..., 1.928
Accelerometer-2,1.980, 1.954, ..., 2.045
Accelerometer-1,1.915, 1.954, ..., 2.006END
Accelerometer-2,2.096, 2.122, ..., 2.109END
```

Because the sensor is identified in every string (by "Accelerometer-1" and "Accelerometer-2"), these tags may be used as identifiers to concatenate the arrays together. The use of the "END" identifier ensures that the final string properly flags the concatenation process to stop so a new array can be captured.

The Arduino microcontroller is programmed to follow the flow diagram pictured in Figure A.4. While no signal is being transmitted to the device, it waits for either a *Set* or

Figure A.5: Node Red startup flow

Sample command. The *Set* command syntax is designed to allow for some flexibility in the microcontroller parameters so that the embedded code does not need to be updated for minor timing changes. When the *Sample* command is received, the microcontroller enters a sampling loop in which it will perform ADC operations on all of the connected sensors. These converted digital signals are appended to their respective arrays until the sampling interval is complete. At this time, all of the stored sensor data is transmitted via the UART interface to the Beaglebone computer and the memory is cleared. After waiting for *Dwell Time*, the microcontroller will either restart the loop or enter the *Wait for Command* state based on whether the *Sample* command is still being received.

A.1.3 Embedded Linux Device Software

While the Teensy microcontroller is sampling and transmitting sensor data, the Beaglebone Black embedded Linux device receives, parses, and prepares the data for transmission through the digital architecture. The open source Node Red tool serves as an excellent platform to perform these tasks. Upon powering up, the Beaglebone Black opens an instance of Node Red and configures the startup parameters as shown in Figure A.5. Default global variables for the microcontroller such as *Dwell Time*, *Sampling Interval*, and number of samples N are created. In addition, the UART pins are configured to allow communication between the Beaglebone and the Teensy. After this, a *Sample* command is transmitted to begin the sampling process.

The main Node Red flow on the Beaglebone Black is pictured in Figure A.6. A *Serial In* node is configured to receive serial data from the Teensy microcontroller. A timestamp is then attached to the input string. Because this microcontroller lacks the network connectivity to update an internal clock, it is necessary to add this timestamp from the Beaglebone Black instead. Next, the Node Red code begins concatenating sensor

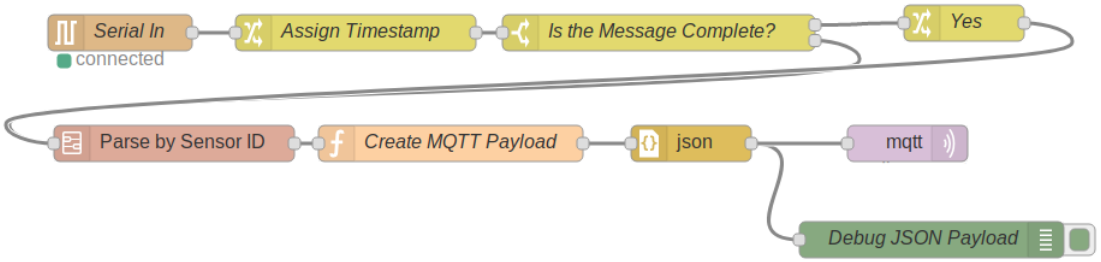


Figure A.6: Node Red main flow

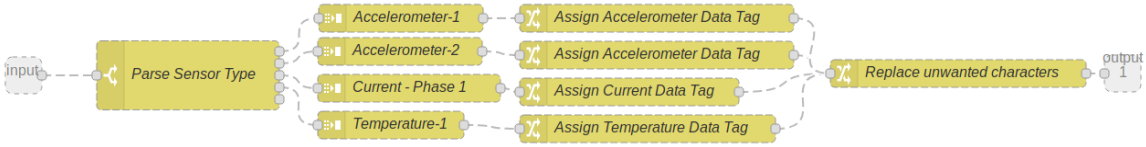


Figure A.7: Node Red sensor-specific flow

strings which were transmitted in multiple lines. Once an "END" has been detected, specific instructions based on the exact sensor are followed in the "Parse by Sensor ID" subflow. Once the sensor data have been properly categorized, a Javascript function is used to format the data into an MQTT-compatible payload. This payload is then transmitted to the MQTT message broker.

The sensor-specific subflow is shown in Figure A.7. In this configuration, the Node Red is designed to accommodate two accelerometers, one current sensor, and one temperature sensor. Based on rules established in Section A.1.2, the UART string is properly associated with its corresponding sensor type and parsed as an array object.

REFERENCES

[1] S. W. Zeng, "Discussion on maintenance strategy, policy and corresponding maintenance systems in manufacturing," *Reliability Engineering & System Safety*, vol. 55, no. 2, pp. 151–162, Feb. 1997.

[2] H. M. Hashemian, "State-of-the-Art Predictive Maintenance Techniques," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 1, pp. 226–236, Jan. 2011.

[3] J.-H. Shin and H.-B. Jun, "On condition based maintenance policy," *Journal of Computational Design and Engineering*, vol. 2, no. 2, pp. 119–127, Apr. 2015.

[4] R. Gao, L. Wang, R. Teti, D. Dornfeld, S. Kumara, M. Mori, and M. Helu, "Cloud-enabled prognosis for manufacturing," *CIRP Annals*, vol. 64, no. 2, pp. 749–772, 2015.

[5] J. Lee, E. Lapira, B. Bagheri, and H.-a. Kao, "Recent advances and trends in predictive manufacturing systems in big data environment," *Manufacturing Letters*, vol. 1, no. 1, pp. 38–41, Oct. 2013.

[6] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, "How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective," vol. 8, no. 1, p. 8, 2014.

[7] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir, and T. Eschert, "Industrial Internet of Things and Cyber Manufacturing Systems," in *Industrial Internet of Things*, S. Jeschke, C. Brecher, H. Song, and D. B. Rawat, Eds., Cham: Springer International Publishing, 2017, pp. 3–19, ISBN: 978-3-319-42558-0 978-3-319-42559-7.

[8] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, Jun. 2017.

[9] D. Singh, G. Tripathi, and A. J. Jara, "A survey of Internet-of-Things: Future vision, architecture, challenges and services," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Mar. 2014, pp. 287–292.

[10] "MTConnect," *mtconnect.org*, 2008.

[11] "OPC-UA," *https://opcfoundation.org/about/opc-technologies/opc-ua/*, 2008.

[12] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, Jan. 2015.

[13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[14] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges," in *2012 10th International Conference on Frontiers of Information Technology*, Islamabad, Pakistan: IEEE, Dec. 2012, pp. 257–260, ISBN: 978-0-7695-4927-9 978-1-4673-4946-8.

[15] Z. Bi, L. D. Xu, and C. Wang, "Internet of Things for Enterprise Systems of Modern Manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1537–1546, May 2014.

[16] X. Li, D. Li, J. Wan, A. V. Vasilakos, C.-F. Lai, and S. Wang, "A review of industrial wireless networks in the context of Industry 4.0," *Wireless Networks*, vol. 23, no. 1, pp. 23–41, Jan. 2017.

[17] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, "A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1457–1477, 2017.

[18] D. Serpanos and M. Wolf, "Industrial internet of things," in *Internet-of-Things (IoT) Systems*, Springer, 2018, pp. 37–54.

[19] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt, "A detailed analysis of iot platform architectures: Concepts, similarities, and differences," in *Internet of Everything*, Springer, 2018, pp. 81–101.

[20] V. Vallois, F. Guenane, and A. Mehaoua, "Reference architectures for security-by-design iot: Comparative study," in *2019 Fifth Conference on Mobile and Secure Services (MobiSecServ)*, IEEE, 2019, pp. 1–6.

[21] H. S. Kang, J. Y. Lee, S. Choi, H. Kim, J. H. Park, J. Y. Son, B. H. Kim, and S. D. Noh, "Smart manufacturing: Past research, present findings, and future directions," *International Journal of Precision Engineering and Manufacturing-Green Technology*, vol. 3, no. 1, pp. 111–128, Jan. 2016.

[22] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of industry 4.0: A review," *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.

[23] J. Qin, Y. Liu, and R. Grosvenor, "A categorical framework of manufacturing for industry 4.0 and beyond," *Procedia Cirp*, vol. 52, pp. 173–178, 2016.

[24] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE international systems engineering symposium (ISSE)*, IEEE, 2017, pp. 1–7.

[25] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, *et al.*, *Extensible markup language (xml) 1.0*, 2000.

[26] D. Crockford, "The json data interchange format," *Technical report, 2013, ECMA International.*, 2013.

[27] R. Lynn, W. Louhichi, M. Parto, E. Wescoat, and T. Kurfess, "Rapidly Deployable MTConnect-Based Machine Tool Monitoring Systems," in *Volume 3: Manufacturing Equipment and Systems*, Los Angeles, California, USA: ASME, Jun. 2017, V003T04A046, ISBN: 978-0-7918-5074-9.

[28] P. D. Urbina Coronado, R. Lynn, W. Louhichi, M. Parto, E. Wescoat, and T. Kurfess, "Part data integration in the Shop Floor Digital Twin: Mobile and cloud technologies to enable a manufacturing execution system," *Journal of Manufacturing Systems*, vol. 48, pp. 25–33, Jul. 2018.

[29] D. Wu, S. Liu, L. Zhang, J. Terpenny, R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," *Journal of Manufacturing Systems*, vol. 43, pp. 25–34, Apr. 2017.

[30] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, Jul. 2018.

[31] P. O'Donovan, C. Gallagher, K. Bruton, and D. T. O'Sullivan, "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications," *Manufacturing Letters*, vol. 15, pp. 139–142, Jan. 2018.

[32] "Siemens 828d controller," *https://assets.new.siemens.com/siemens/assets/api/uuid:ae8a91b5-48ce* accessed: 2020-07-09.

[33] M. George, Akash J B, A. Hussain, and S. P. Sreelal, "A compact multichannel data acquisition and processing system for IoT applications," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Kochi, India: IEEE, Aug. 2015, pp. 1263–1267, ISBN: 978-1-4799-8790-0 978-1-4799-8792-4.

[34] R. Lynn, E. Wescoat, D. Han, and T. Kurfess, "Embedded fog computing for high-frequency MTConnect data analytics," *Manufacturing Letters*, vol. 15, pp. 135–138, Jan. 2018.

[35] K. S. Saleeby and T. Kurfess, "Low cost wireless accelerometer sensor platform with internet-of-things for manufacturing (IOT4mfg) applications," in *Micro- and Nanotechnology Sensors, Systems, and Applications XI*, M. S. Islam and T. George, Eds., Baltimore, United States: SPIE, May 2019, p. 35, ISBN: 978-1-5106-2629-4 978-1-5106-2630-0.

[36] B. Bengherbia, M. Ould Zmirli, A. Toubal, and A. Guessoum, "FPGA-based wireless sensor nodes for vibration monitoring system and fault diagnosis," *Measurement*, vol. 101, pp. 81–92, Apr. 2017.

[37] J. Gutiérrez, J. F. Villa-Medina, A. Nieto-Garibay, and M. Porta-Gándara, "Automated Irrigation System Using a Wireless Sensor Network and GPRS Module," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 1, pp. 166–176, Jan. 2014.

[38] L. Hou and N. W. Bergmann, "Novel Industrial Wireless Sensor Networks for Machine Condition Monitoring and Fault Diagnosis," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 10, pp. 2787–2798, Oct. 2012.

[39] "Nist sms testbed: Volatile data stream," *https://smstestbed.nist.gov/vds/sample*, accessed: 2020-04-22.

[40] "Opc ua companion specification - mtconnect," *https://www.mtconnect.org/opc-ua-companion-specification/*, accessed: 2020-04-27.

[41] "Automated test and automated measurement systems - national instruments," *https://www.ni.com/en-us.html*, accessed: 2020-05-05.

[42] "Ncd manufactures plug and play hardware for iot industrial automation," *https://ncd.io/*, accessed: 2020-05-04.

[43] R. Ahmad and S. Kamaruddin, "An overview of time-based and condition-based maintenance in industrial application," *Computers & industrial engineering*, vol. 63, no. 1, pp. 135–149, 2012.

[44] D. Wang, K.-L. Tsui, and Q. Miao, "Prognostics and health management: A review of vibration based bearing and gear health indicators," *IEEE Access*, vol. 6, pp. 665–676, 2017.

[45] N Tandon and A Choudhury, "A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings," *Tribology International*, vol. 32, no. 8, pp. 469–480, Aug. 1999.

[46] A. Rai and S. Upadhyay, "A review on signal processing techniques utilized in the fault diagnosis of rolling element bearings," *Tribology International*, vol. 96, pp. 289–306, Apr. 2016.

[47] G. White, *Introduction to Machine Vibration Monitoring*. DLI Engineering Corp, 1997.

[48] R. B. Randall, *Vibration-based condition monitoring: industrial, aerospace and automotive applications*. John Wiley & Sons, 2011.

[49] D. Wang, K.-L. Tsui, and Y. Qin, "Optimization of segmentation fragments in empirical wavelet transform and its applications to extracting industrial bearing fault features," *Measurement*, vol. 133, pp. 328–340, 2019.

[50] A. M. Al-Ghamd and D. Mba, "A comparative experimental study on the use of acoustic emission and vibration analysis for bearing defect identification and estimation of defect size," *Mechanical systems and signal processing*, vol. 20, no. 7, pp. 1537–1571, 2006.

[51] A. Chen and T. R. Kurfess, "A new model for rolling element bearing defect size estimation," *Measurement*, vol. 114, pp. 144–149, 2018.

[52] J. Antoni, "The spectral kurtosis: A useful tool for characterising non-stationary signals," *Mechanical systems and signal processing*, vol. 20, no. 2, pp. 282–307, 2006.

[53] Y. Lei, N. Li, S. Gontarz, J. Lin, S. Radkowski, and J. Dybala, "A model-based method for remaining useful life prediction of machinery," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1314–1326, 2016.

[54] T. Benkedjouh, K. Medjaher, N. Zerhouni, and S. Rechak, "Remaining useful life estimation based on nonlinear feature reduction and support vector regression," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 7, pp. 1751–1760, 2013.

[55] P. Welch, "The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[56] D. Dyer and R. M. Stewart, "Detection of Rolling Element Bearing Damage by Statistical Vibration Analysis," *Journal of Mechanical Design*, vol. 100, no. 2, p. 229, 1978.

[57] N. Li, Y. Lei, J. Lin, and S. X. Ding, "An Improved Exponential Model for Predicting Remaining Useful Life of Rolling Element Bearings," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7762–7773, Dec. 2015.

[58] C. J. Li and J. Ma, "Wavelet decomposition of vibrations for detection of bearing-localized defects," *Ndt & E International*, vol. 30, no. 3, pp. 143–149, 1997.

[59] K Mori, N Kasashima, T Yoshioka, and Y Ueno, "Prediction of spalling on a ball bearing by applying the discrete wavelet transform to vibration signals," *Wear*, vol. 195, no. 1-2, pp. 162–168, 1996.

[60] N Baydar and A. Ball, "Detection of gear failures via vibration and acoustic signals using wavelet transform," *Mechanical Systems and Signal Processing*, vol. 17, no. 4, pp. 787–804, 2003.

[61] W. Wang and P. McFadden, "Application of wavelets to gearbox vibration signals for fault detection," *Journal of sound and vibration*, vol. 192, no. 5, pp. 927–939, 1996.

[62] J. Antoni and R. Randall, "The spectral kurtosis: Application to the vibratory surveillance and diagnostics of rotating machines," *Mechanical systems and signal processing*, vol. 20, no. 2, pp. 308–331, 2006.

[63] G. Y. J. L. J. Lee H. Qiu and R. T. Services, "Ims, university of cincinnati. "bearing data set" nasa ames prognostics data repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), nasa ames research center, moffett field, ca,"

[64] A. Agogino and K. Goebel, ""milling data set ", nasa ames prognostics data repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), nasa ames research center, moffett field, ca,"

[65] Ö. F. Eker, F. Camci, and I. K. Jennions, "Major challenges in prognostics: Study on benchmarking prognostic datasets," 2012.

[66] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *2016 Cloudification of the Internet of Things (CIoT)*, Paris, France: IEEE, Nov. 2016, pp. 1–6, ISBN: 978-1-5090-4960-8.

[67] "Messages - iotfm architecture," *https://arch.fis.gatech.edu/messages/*, accessed: 2020-04-06.

[68] I Yesilyurt and H Ozturk, "Tool condition monitoring in milling using vibration analysis," *International journal of production research*, vol. 45, no. 4, pp. 1013–1028, 2007.

[69] P. Sivasakthivel, V Velmurugan, and R Sudhakaran, "Prediction of vibration amplitude from machining parameters by response surface methodology in end milling," *The International Journal of Advanced Manufacturing Technology*, vol. 53, no. 5-8, pp. 453–461, 2011.

[70] C. Drouillet, J. Karandikar, C. Nath, A.-C. Journeaux, M. El Mansori, and T. Kurfess, "Tool life predictions in milling using spindle power with the neural network technique," *Journal of Manufacturing Processes*, vol. 22, pp. 161–168, 2016.

[71] "Python," *https://www.python.org/*, accessed: 2020-04-24.

[72] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[73] A Gulli, A Kapoor, and S Pal, *Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API*. Packt Publishing, 2019.

[74] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly, 2019.

[75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[76] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey,
bibinitperiodI. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[77] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[78] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[79] M. Cerrada, R.-V. Sánchez, C. Li, F. Pacheco, D. Cabrera, J. Valente de Oliveira, and R. E. Vásquez, "A review on data-driven fault severity assessment in rolling bearings," *Mechanical Systems and Signal Processing*, vol. 99, pp. 169–196, Jan. 2018.

[80] P. Kankar, S. C. Sharma, and S. Harsha, "Fault diagnosis of ball bearings using continuous wavelet transform," *Applied Soft Computing*, vol. 11, no. 2, pp. 2300–2312, Mar. 2011.

[81] J. Ben Ali, N. Fnaiech, L. Saidi, B. Chebel-Morello, and F. Fnaiech, "Application of empirical mode decomposition and artificial neural network for automatic bearing fault diagnosis based on vibration signals," *Applied Acoustics*, vol. 89, pp. 16–27, Mar. 2015.

[82] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[83] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, Jan. 2019.

[84] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[85] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models," *arXiv:1708.08296 [cs, stat]*, Aug. 2017, arXiv: 1708.08296.

[86] M. Markou and S. Singh, "Novelty detection: A review—part 1: Statistical approaches," *Signal processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[87] M. L. George, J. Maxey, D. T. Rowlands, and M. Upton, *Lean six sigma pocket toolbook*. McGraw-Hill Professional Publishing, 2004.

[88] L. S. Nelson, "The shewhart control chart—tests for special causes," *Journal of quality technology*, vol. 16, no. 4, pp. 237–239, 1984.

[89] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, 2001, pp. 41–46.

[90] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA: 2015, vol. 2018.

[91] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[92] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Computer Communications*, vol. 89-90, pp. 5–16, Sep. 2016.

[93] "Node-red," *https://nodered.org/*, accessed: 2020-05-04.

[94] *BeagleBoard.org - community supported open hardware computers for making*.

[95] "5 axis desktop cnc milling machines," *https://pocketnc.com/*, accessed: 2020-05-12.

[96] "Creator," *https://www.creator.rest/*, accessed: 2020-05-12.

[97] "Chai — molecular testing for everything," *https://www.chaibio.com/*, accessed: 2020-05-12.

[98] S. Lee, H. Kim, D.-k. Hong, and H. Ju, "Correlation analysis of mqtt loss and delay according to qos level," in *The International Conference on Information Networking 2013 (ICOIN)*, IEEE, 2013, pp. 714–717.